
VMS Version 5.5 New Features Manual

Order Number: AA-LA97D-TE

November 1991

This manual describes the new features of the VMS Version 5.5 operating system. It also describes new features from past VMS releases that have not been documented in other printed manuals.

Revision/Update Information: This manual supersedes the *VMS Version 5.4 New Features Manual*.

Software Version: VMS Version 5.5

**Digital Equipment Corporation
Maynard, Massachusetts**

November 1991

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

© Digital Equipment Corporation 1991.

All Rights Reserved.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: BI, CDA, CI, DBMS, DDIF, DECdtm, DECnet, DECwindows, Digital, DSSI, DBMS, HSC, LAT, MASSBUS, MicroVAX, MSCP, Q-22 bus, RA, Rdb/VMS, TMSCP, UETP, UNIBUS, VAX, VAX Ada, VAXBI, VAXcluster, VAX DOCUMENT, VAX FORTRAN, VAX MACRO, VAXstation, VAX Volume Shadowing, VMS, and the DIGITAL logo.

The following are third-party trademarks:

PostScript is a registered trademark of Adobe Systems Incorporated.

UNIX is a registered trademark of UNIX System Laboratories, Inc.

ZK5719

This document was prepared using VAX DOCUMENT, Version 1.2

Contents

Preface	xvii
Part I Summary of New Features	
1 Summary of New VMS Version 5.5 Features	
1.1 New Software Features	1-1
1.2 Announcing the New VMS Dependability Handbook	1-4
1.3 Announcing the New BACKUP Utility Guide	1-4
Part II General User Features	
2 DCL Commands and Lexical Functions	
2.1 DCL Command Enhancements	2-2
INITIALIZE/SIZE	2-3
SET VOLUME/REBUILD[=FORCE]	2-4
SET MAGTAPE/RETENSION	2-5
CONVERT/DOCUMENT/MESSAGE_FILE=filespec	2-6
2.2 F\$MESSAGE Lexical Function	2-7
3 Batch and Print Queuing System	
3.1 Changes to SHOW ENTRY	3-1
3.1.1 Change in Format of SHOW ENTRY Display	3-1
3.1.2 SHOW ENTRY Command Accepts Job Names	3-1
3.1.3 New Stalled Job State	3-2
3.2 Change in Format of SHOW QUEUE Display	3-2
3.3 User-Specified Job Retention	3-3
3.3.1 Uses for User-Specified Job Retention	3-3
3.3.2 Job Retention Command Syntax	3-3
3.3.3 How Job Retention Is Determined	3-4
3.3.4 Timed Retention	3-5
3.4 Batch Log Time-Stamps	3-5
3.4.1 New DCL Command: SET PREFIX	3-6
3.4.2 New Item for F\$ENVIRONMENT Lexical Function	3-7
3.5 /NOTE Qualifier for SUBMIT Command	3-8
3.6 Changes to F\$GETQUI Lexical Function	3-8

4 VMS System Messages

Part III System Management Features

5 VMS Batch and Print Queuing System

5.1	Clusterwide Queue Manager	5-1
5.2	New Queue Database Design	5-2
5.2.1	Moving Queue Database Files from Their Default Location	5-4
5.2.1.1	Moving the Master File	5-4
5.2.1.2	Moving the Queue and Journal Files	5-4
5.3	Starting and Stopping the Queue Manager	5-5
5.3.1	Starting the Queue Manager	5-5
5.3.1.1	Customizing Queue Manager Failover	5-6
5.3.1.2	Automatic Queue Manager Restart	5-6
5.3.1.3	If the Queue Manager Is Already Started	5-6
5.3.1.4	Obsolete Qualifiers	5-7
5.3.2	Stopping the Queue Manager	5-7
5.3.3	Stopping Queues on a Node	5-7
5.4	The Autostart Feature	5-7
5.4.1	Designating a Queue as an Autostart Queue	5-8
5.4.1.1	Setting Up Autostart Queues for Automatic Failover	5-8
5.4.2	Enabling Autostart on a Node	5-9
5.4.3	Starting Autostart Queues	5-9
5.4.4	Preventing Autostart Queues from Starting	5-9
5.4.5	Disabling Autostart on a Node	5-10

6 LADCP Utility

7 Clusterwide Tape Serving

7.1	Loading the Magnetic Tape Server	7-1
7.1.1	TMSCP_LOAD Parameter	7-1
7.1.2	TAPE_ALLOCLASS Parameter	7-3

8 VMS Volume Shadowing Phase II Enhancements

8.1	Specifying the Shadow Set Member Recovery Timeout Period	8-1
8.2	Volume Shadowing Phase II Supports Digital SCSI Devices	8-2

9 LAT New Features

9.1	Starting Up the LAT Protocol Software	9-1
9.2	Site-Specific LAT Command Procedure (LAT\$SYSTARTUP.COM)	9-3
9.2.1	Creating a VMS Service	9-4
9.2.2	Setting Up Ports	9-4
9.2.3	Enabling Outgoing LAT Connections	9-5
9.3	Connecting to a LAT Network	9-5
9.3.1	Function of the LAT Protocol Software	9-5
9.3.2	Advantages of the LAT Protocol Software	9-6

9.3.3	The LAT Network	9-6
9.3.3.1	VMS Service Nodes	9-7
9.3.3.2	Terminal Server Nodes	9-8
9.3.3.3	VMS Nodes Allowing Incoming and Outgoing Connections	9-9
9.3.3.4	Using the SET HOST/LAT Command	9-9
9.3.3.5	A Sample LAT Configuration	9-11
9.3.3.6	LAT Relationship to VMS Clusters and DECnet	9-12
9.3.4	Summary of LAT System Management Tasks	9-12
9.3.4.1	Starting Up the LAT Protocol Software	9-13
9.3.4.2	Customizing LAT Characteristics	9-13
9.3.4.3	Using LATCP to Control the LAT Protocol Software	9-13
9.3.4.4	Managing the LATACP Database Size	9-14

10 VMS License Management Facility

10.1	Moving and Copying Licenses	10-1
10.2	Deleting Licenses	10-1
10.3	Automating License Registration	10-1
10.4	Creating License Reservation Lists	10-1
10.5	Support for PAKs with the RESERVE_UNITS Option	10-2
10.6	Ease-of-Use Features	10-2
10.7	Revised SYS\$UPDATE:VMSLICENSE.COM	10-2

11 Movefile Command Qualifiers

11.1	SET FILE Command Qualifiers	11-1
	SET FILE/NOMOVE [/MOVE]	11-2
11.2	DIRECTORY/FULL, DUMP/HEADER, and DUMP/FILE_HEADER Commands	11-3
11.3	Critical System Files	11-4

Part IV Programming Features

12 System Service Support for the VMS Batch and Print Queuing System

12.1	\$GETQUI and \$SNDJBC System Services	12-1
12.1.1	\$GETQUI Service	12-1
12.1.2	\$SNDJBC Service	12-2

13 Run-Time Library Routines

13.1	LIB\$GETQUI Run-Time Library Routine	13-1
13.2	Fast-Vector Math Routines	13-1
13.2.1	Exception Handling	13-2
13.2.2	Special Restrictions on Input Arguments	13-2
13.2.3	Accuracy	13-3
13.2.4	Performance	13-3
13.3	Parallel Processing Routines	13-3
13.3.1	Enhancements for Unique Naming	13-3
13.3.2	Spin/Wait Options for Blocking Synchronization	13-4

14	VMS Debugger: Tasking and Multithread Support	
14.1	Command Interface: Enhanced Commands and Qualifiers	14-1
14.2	DECwindows Interface: Enhancements	14-1
15	DECthreads	
15.1	Overview	15-1
16	DECdtm System Services: New and Changed Features	
16.1	Abort Reason Codes	16-1
16.2	Transaction Timeouts	16-2
16.3	New and Modified System Dump Analyzer Commands	16-2
17	LAT \$QIO Functions	
17.1	LAT SETMODE \$QIO Function	17-1
17.2	LAT SENSEMODE \$QIO Function	17-7
18	Asynchronous Printer Support	
	SET TERMINAL/COMMSYNC/NOCOMMSYNC	18-2
19	Support for Case Sensitivity	
19.1	Linker Support for Case-Sensitive Languages	19-1
19.2	VAX MACRO Support for Case Sensitivity	19-2
19.2.1	MACRO Programs That Reference Other MACRO Modules	19-3
19.2.2	MACRO Programs That Reference the Same MACRO Module	19-4
19.2.3	Uppercase Languages to MACRO Programs	19-4
19.2.4	Lowercase Languages to MACRO Programs	19-4
19.2.4.1	MACRO Command /NAMES Qualifier	19-5
	MACRO/NAMES	19-6
20	System Dump Analyzer	
20.1	TMSCP Symbol	20-1
20.2	Support for Transaction Processing	20-1
	SHOW LOGS	20-2
	SHOW PROCESS/PARTICIPANTS	20-3
	SHOW PROCESS/TRANSACTIONS	20-4
	SHOW TRANSACTIONS	20-6
21	Mailbox Driver	
21.1	Unidirectional Mailboxes	21-1
21.2	Mailbox Driver Functions and Modifiers	21-1
21.2.1	Wait for Writer/Reader Function	21-1
21.2.2	IO\$M_WRITERCHECK Function Modifier	21-2
21.2.3	IO\$M_READERCHECK Function Modifier	21-2
21.2.4	IO\$M_STREAM Function Modifier	21-2

22 \$QIO Support for Moving Disk Files

22.1	Calling the Movefile Subfunction	22-1
22.1.1	Input Parameters	22-1
22.1.2	Operation	22-3

A VMS Version 5.4-3 Features

A.1	Summary of New VMS Version 5.4-3 Software Features	A-1
A.2	VMS Version 5.4-3 System Management Features	A-2
A.2.1	Backup Utility	A-2
A.2.1.1	/RELEASE_TAPE Qualifier	A-2
A.2.1.2	ACCESSIBILITY Keyword	A-3
A.2.1.3	Backup Label Processing Options	A-3
A.2.2	Disk and Tape Class Drivers—Enhanced Error Reporting	A-3
A.2.3	New NCP Line Counters for FDDI Communications	A-4
A.2.4	FDDI/Ethernet Startup Error Code	A-5
A.2.5	Proactive Reclamation of Memory from Idle Processes	A-5
A.2.5.1	How Is This Policy Enabled?	A-6
A.2.5.2	Reclaiming Memory from Long-Waiting Processes	A-6
A.2.5.3	Reclaiming Memory from Periodically Waking Processes	A-7
A.2.5.3.1	Setting the FREEGOAL Parameter	A-7
A.2.5.3.2	Sizing Page and Swap Files	A-7
A.2.6	Tape Support	A-8
A.2.7	VMSINSTAL Callback RUN_IMAGE: New Parameter	A-8
A.3	VMS Version 5.4-3 Programming Features	A-8
A.3.1	Open-Bus Driver Support Features	A-8
A.3.1.1	VMS Device Support for VMEbus Devices	A-8
A.3.1.1.1	Hardware Environment	A-9
A.3.1.1.2	Associated Documents	A-9
A.3.1.1.3	Selecting VMEbus Protocol Parameters	A-10
A.3.1.1.4	Considering Byte-Order Transfer Differences	A-10
A.3.1.1.5	Handling Interrupts	A-11
A.3.1.1.6	DMA Operations	A-12
A.3.1.1.7	Programmed I/O Operations and I/O Mapping	A-13
A.3.1.1.8	Coding a VMEbus Device Driver	A-15
A.3.1.1.9	Assembling and Linking a VMEbus Driver	A-17
A.3.1.1.10	Loading a VME Device Driver	A-17
A.3.1.1.11	VMS Macros Invoked by VME Drivers	A-18
	SWAPLONG	A-19
	SWAPWORD	A-20
A.3.1.2	VME Driver Operating System Routines	A-21
	IOC\$ALOVMEMAP_DMA, IOC\$ALOVMEMAP_DMAN	A-22
	IOC\$LOADVMEMAP_DMA, IOC\$LOADVMEMAP_DMAN	A-24
	IOC\$RELVMEMAP_DMA, IOC\$RELVMEMAP_DMAN	A-26
	IOC\$ALOVMEMAP_PIO	A-28
	IOC\$LOADVMEMAP_PIO	A-29
	IOC\$RELVMEMAP_PIO	A-31
	IOC\$VME_BYTE_SWAP_LONG	A-33
	IOC\$VME_BYTE_SWAP_WORD	A-34
A.3.1.3	Sample Driver for a VMEbus DR11-W	A-35

A.3.2	SCSI Device Support for the NCR 53C94 Controller	A-51
A.3.2.1	SCSI Device Driver Data Structures	A-51
A.3.2.2	Using the SPI\$CONNECT Macro and Maximum Byte Counts ...	A-51
A.3.3	FDDI and Ethernet—VMS Support	A-52
A.3.3.1	Overview of FDDI	A-52
A.3.3.2	New FDDI Device DEMFA	A-52
A.3.3.3	Programming Interface	A-52
A.3.3.4	Parameters	A-54
A.3.3.4.1	NMA\$C_PCLI_MED (Medium)	A-54
A.3.3.4.2	NMA\$C_PCLI_RFC (Receive Frame Control)	A-54
A.3.3.4.3	NMA\$C_PCLI_XFC (Transmit Frame Control)	A-55
A.3.3.4.4	NMA\$C_PCLI_BUS (Maximum Receive Buffer Size)	A-55
A.3.3.4.5	NMA\$C_PCLI_MBS (Maximum Packet Length)	A-55
A.3.3.4.6	NMA\$C_PCLI_CCA (Can Change Address)	A-55
A.3.3.5	Frame and Packet Formats	A-56
A.3.3.5.1	FDDI Frames	A-56
A.3.3.5.2	CSMA/CD Frames	A-57
A.3.3.5.3	Packet Formats	A-58
A.3.4	Preferred Access Path Programming Examples	A-60
A.3.5	VAX Ada Run-Time Library	A-60
A.3.6	DECwindows X11 Display Server—Color Name File	A-62
A.3.7	Changes to SDA SHOW PORTS Command	A-62

B VMS Version 5.4 Features

B.1	Summary of New VMS Version 5.4 Software Features	B-1
B.2	Introduction to Vector Processing	B-4
B.2.1	Overview of the Vector Processing Environment	B-5
B.2.1.1	VAX Vector Processing Systems	B-5
B.2.1.2	Vectorized Programs	B-7
B.2.1.3	VMS Support for Vector Processing	B-8
B.2.1.3.1	Life of a Vector Consumer	B-8
B.2.1.3.2	VAX Vector Instruction Emulation Facility (VVIEF)	B-10
B.2.2	Managing the Vector Processing Environment	B-10
B.2.2.1	Loading the VMS Vector Processing Support Code	B-11
B.2.2.2	Configuring a VMS Vector Processing System	B-11
B.2.2.3	Managing Vector Processes	B-12
B.2.2.3.1	Adjusting System Resources and Process Quotas	B-12
B.2.2.3.2	Distributing Scalar and Vector Resources Among Processes ..	B-13
B.2.2.4	Restricting Access to the Vector Processor by Using ACLs	B-13
B.2.2.5	Obtaining Information About a Vector Processing System	B-14
B.2.2.5.1	DCL Lexical Functions F\$GETJPI and F\$GETSYI	B-14
B.2.2.5.2	SHOW CPU Command	B-15
B.2.2.5.3	SHOW PROCESS and LOGOUT/FULL Commands	B-15
B.2.2.5.4	Vector Processing Support Within the VMS Accounting Utility (ACCOUNTING)	B-15
B.2.2.5.5	Vector Support Within the Error Log Utility (ERROR LOG)	B-16
B.2.2.5.6	Vector Support Within the VMS Monitor Utility (MONITOR)	B-16
B.2.2.6	Loading the VAX Vector Instruction Emulation Facility (VVIEF)	B-16
B.2.2.7	System Messages Related to Vector Processing Activities	B-17

B.2.3	Programming in a Vector Processing Environment	B-21
B.2.3.1	Vector Routines in the MTH\$ Run-Time Library	B-23
B.2.3.2	Obtaining Information About a Vector Processing System	B-24
B.2.3.3	Releasing the Vector Processor	B-24
B.2.3.4	Preserving and Restoring a Routine's Vector State	B-25
B.2.3.5	Debugging a Vectorized Program	B-26
B.2.3.5.1	Vector Processing Support Within the VMS Debugger	B-26
B.2.3.5.2	Vector Processing Support Within the VMS System Dump Analyzer (SDA)	B-27
B.2.3.5.3	Vector Processing Support Within the VMS Delta/XDelta Utility	B-27
B.2.3.5.4	Vector Processing Support Within the VMS Patch Utility	B-28
B.2.3.6	Servicing Vector Exceptions	B-28
B.2.3.7	Requirements of the VAX Procedure Calling and Condition Handling Standard	B-31
B.2.3.7.1	Vector Register Usage	B-32
B.2.3.7.2	Vector and Scalar Processor Synchronization	B-32
B.2.3.7.3	Memory Synchronization	B-32
B.2.3.7.4	Exception Synchronization	B-32
B.2.3.7.5	Synchronization Summary	B-33
B.2.3.7.6	Condition Handler Parameters and Invocation	B-33
B.2.3.8	VMS Accounting Utility (ACCOUNTING) Resource Packet Format	B-33
B.2.3.9	VMS Monitor Utility (MONITOR) VECTOR Class Record	B-33
B.3	Introduction to DECdtm Services	B-34
B.3.1	Characteristics of Distributed Transactions	B-34
B.3.2	Transaction Processing System Model	B-36
B.3.2.1	Resource Manager	B-36
B.3.2.2	Transaction Manager	B-36
B.3.2.3	Log Manager	B-38
B.3.3	Overview of Two-Phase Commit Protocol	B-39
B.3.4	Managing DECdtm Services Using VMS Utilities	B-40
B.3.5	New TRANSACTION_ID Data Type for Programming Routines	B-40
B.4	VMS Version 5.4 General User Features	B-40
B.4.1	DCL Commands	B-41
	BACKUP/MEDIA_FORMAT=[NO]COMPACTION	B-42
	MOUNT/MEDIA_FORMAT=[NO]COMPACTION	B-43
B.4.2	System Messages	B-44
B.4.2.1	System Messages Available from Online Help	B-44
B.4.3	DECwindows User and Desktop Applications	B-45
B.4.3.1	Session Manager	B-45
B.4.3.2	Setting Another Session Language	B-45
B.4.3.3	Changing Your Target Screen	B-45
B.4.3.4	CDA Viewer	B-46
B.4.3.5	Viewing a PostScript File	B-46
B.4.3.6	New Processing Options for Viewing PostScript Files	B-47
B.4.4	Calculator	B-48
B.4.5	Clock	B-48
B.4.6	Mail: Displaying PostScript Files	B-48
B.5	VMS Version 5.4 System Management Features	B-49
B.6	AUTOGEN Command Procedure	B-49
B.6.1	Parameter Name Validation	B-49
B.6.2	AGEN\$FEEDBACK.REPORT Replaced by New File	B-50
B.6.3	MODPARAMS.DAT Includes External Parameter Files	B-50

B.6.4	MIN_, MAX_, and ADD_ Values Allowed for Page and Swap Files . . .	B-51
B.6.5	New Feedback Parameters	B-52
B.6.6	Logical Names Defined by AUTOGEN	B-52
B.6.7	New Technique for Running AUTOGEN in Batch Mode	B-52
B.6.8	Using MAIL to Send AGEN\$PARAMS.REPORT	B-54
B.7	VAXcluster Management	B-55
B.7.1	CI Architecture Extensions	B-55
B.7.2	MSCP Server Load Sharing	B-55
B.7.3	Preferred Path Support for DSA disks	B-55
B.8	System Generation Utility (SYSGEN)	B-56
B.8.1	SCSI_NOAUTO Parameter	B-56
B.8.2	LOAD_PWD_POLICY Parameter	B-57
B.8.3	LOAD_SYS_IMAGES Parameter	B-57
B.8.4	Supported Device Names for VAXft 3000 Systems	B-57
B.8.5	New SYSGEN Commands	B-58
	SHOW/BI=BIindex	B-59
	SHOW/BUS=busId	B-60
	SHOW/XMI=BIindex	B-61
B.9	Error Log Utility (ERROR LOG)	B-62
B.9.1	Supported Device Types for VAXft 3000 Systems	B-62
B.9.2	New Keywords for /EXCLUDE and /INCLUDE Qualifiers	B-62
B.9.3	New Qualifier: /NODE	B-62
	ERROR LOG/NODE	B-63
B.10	System Security	B-64
B.10.1	Site-Defined Password Policy	B-64
B.10.1.1	Screening New Passwords	B-64
B.10.1.1.1	Password History List	B-64
B.10.1.1.2	Site-Specific Filter	B-65
B.10.1.2	Specifying a Password Algorithm	B-65
B.11	Log Manager Control Program Utility (LMCP)	B-66
B.11.1	Managing Transaction Log Files	B-66
B.11.1.1	Defining SYS\$JOURNAL	B-67
B.11.1.2	Placing a Transaction Log File	B-67
B.11.1.3	VAXcluster Failover	B-68
B.11.1.4	Determining Transaction Log File Size	B-69
B.11.1.5	Creating Transaction Log Files	B-69
B.11.1.6	Example of Creating a Transaction Log File	B-70
B.11.1.7	Resizing and Moving Transaction Log Files	B-72
B.11.2	Format of Transaction Log Files	B-73
LMCP Usage Summary		B-75
LMCP Commands		B-76
	CONVERT	B-77
	CREATE	B-78
	DUMP	B-80
	HELP	B-84
	REPAIR	B-85
	ABORT	B-87
	COMMIT	B-88
	EXIT	B-89
	FORGET	B-90
	HELP	B-91

	NEXT	B-92
	SHOW	B-93
B.12	Monitor Utility (MONITOR)	B-95
B.12.1	MONITOR TRANSACTION Command	B-95
	MONITOR TRANSACTION	B-96
B.12.2	TRANSACTION Class Record	B-99
B.12.3	MONITOR VECTOR Command	B-100
	MONITOR VECTOR	B-102
B.12.4	VECTOR Class Record	B-104
B.13	Network Control Program Utility (NCP)	B-105
B.13.1	Line and Circuit Name Support for VAXft 3000 Systems	B-105
B.13.2	Line and Circuit Names for New Ethernet/820 Controllers	B-105
B.14	VMS Volume Shadowing Phase II	B-106
B.15	VMS Version 5.4 Programming Features	B-107
B.16	Larger Page Size Capability with Linker Utility	B-107
	/BPAGE	B-108
B.17	VMS Record Management Services	B-110
B.17.1	VMS RMS Asynchronous Support for Process-Permanent Files	B-110
B.17.2	Local Buffer Maximum Increased	B-110
B.17.3	Access-Mode Protection for VMS RMS	B-111
B.17.3.1	Access-Mode Protected Services	B-111
B.17.3.2	Access-Mode Protected Memory	B-111
B.17.4	Expired-Date Suppression	B-112
B.17.4.1	The Role of XAB\$_NORECORD XABITM	B-112
B.17.4.2	Applications for XAB\$_NORECORD XABITM	B-112
B.18	System Dump Analyzer Utility (SDA)	B-113
B.18.1	New SHOW PROCESS Qualifier: /IMAGES	B-113
B.18.2	New SHOW PROCESS Qualifier: /VECTOR_REGISTERS	B-113
B.19	VMS RMS Journaling: Support for DECdtm Services	B-114
B.19.1	Support for DECdtm Transactions	B-114
B.19.2	RUF Services Emulated	B-114
B.19.3	Network Support	B-114
B.19.4	Record Stream Association	B-115
B.19.4.1	How Streams Become Associated with a Transaction	B-115
B.19.4.2	Stream Association Using RUF and DECdtm Services	B-116
B.19.5	Detached Recovery	B-116
B.19.5.1	Synchronous and Asynchronous Recovery	B-116
B.19.5.2	Partial Recovery	B-117
B.19.6	Placement of Recovery Unit Journals	B-117
B.19.7	Multiple Long-Term Journals Allowed	B-118
B.19.8	Mixed-Version Clusters	B-118

C VMS Version 5.3 Features

C.1	VMS Version 5.3 System Management Features	C-1
C.1.1	Extension of Lock Manager Limit	C-1
C.1.2	NCP Executor Command Changes	C-1
C.1.3	Parameter for SET/DEFINE EXECUTOR	C-1
C.1.4	SHOW EXECUTOR CHARACTERISTICS Command	C-2
C.2	VMS Version 5.3 Support for the VMS Distributed Name Service	C-3
C.2.1	Introduction to the Distributed Name Service	C-3

C.2.2	The DNS Namespace	C-4
C.2.2.1	Planning Namespace Objects	C-4
C.2.2.2	Restrictions	C-4
C.2.2.3	Using the Namespace	C-4
C.2.2.4	Object Names	C-5
C.2.2.5	Object Attributes	C-5
C.2.3	Structure of a Namespace	C-5
C.2.3.1	Naming Syntax	C-6
C.2.3.2	Logical Names	C-7
C.2.3.3	Valid Characters for DNS Names	C-8
C.2.4	Creating Objects	C-9
C.2.5	Modifying Objects	C-11
C.2.6	Distributing the Namespace	C-13
C.2.6.1	Replicating Directories	C-14
C.2.6.2	Types of Directories	C-14
C.2.6.3	Setting Confidence	C-15
C.2.6.4	Maintaining Consistency in Data	C-16
C.2.7	Requesting Information from DNS	C-16
C.2.7.1	Reading Objects	C-17
C.2.7.2	Listing Information	C-20
C.2.7.3	How the Clerk Locates Data	C-23
C.2.8	DNS System Services	C-23
	\$DNS	C-25
	\$DNSW	C-48
C.2.9	DNS Run-Time Routines	C-49
	DNS\$APPEND_SIMPLENAME_TO_RIGHT	C-50
	DNS\$COMPARE_FULLNAME	C-52
	DNS\$COMPARE_SIMPLENAME	C-53
	DNS\$CONCATENATE_NAME	C-54
	DNS\$COUNT_SIMPLENAMES	C-56
	DNS\$CVT_DNSADDRESS_TO_BINARY	C-57
	DNS\$CVT_DNSADDRESS_TO_NODENAME	C-58
	DNS\$CVT_NODENAME_TO_DNSADDRESS	C-60
	DNS\$CVT_TO_USERNAME_STRING	C-62
	DNS\$PARSE_USERNAME_STRING	C-64
	DNS\$REMOVE_FIRST_SET_VALUE	C-67
	DNS\$REMOVE_LEFT_SIMPLENAME	C-69
	DNS\$REMOVE_RIGHT_SIMPLENAME	C-71
C.2.10	Starting the DNS Clerk	C-73
C.2.11	DECnet Event Messages	C-73

D VMS Version 5.2 Features

D.1	VMS Version 5.2 System Management Features	D-1
D.1.1	System Generation Utility (SYSGEN)	D-1
D.1.1.1	DEINSTALL Command Description	D-1
D.1.1.2	ERLBUFFERPAGES Parameter	D-2
D.1.2	NETCONFIG.COM Security Enhancements	D-2
D.1.2.1	Default Access Options	D-2
D.1.2.2	Security Benefits	D-3
D.1.2.3	Questions Posed by NETCONFIG.COM	D-4

D.1.3	New NETCONFIG_UPDATE.COM for Existing Networks	D-4
D.1.3.1	Benefits of NETCONFIG_UPDATE.COM	D-4
D.1.3.2	Using NETCONFIG_UPDATE.COM in a VAXcluster	D-4
D.1.4	Backup Utility (BACKUP)	D-5
D.1.4.1	Performance Enhancements	D-5
D.1.4.2	Setting Up the BACKUP Account	D-5
D.1.4.3	Setting System Generation Utility (SYSGEN) Parameters	D-7
D.1.4.4	Understanding Why the Output Device Seems Idle	D-7
D.1.4.5	/BUFFER_COUNT Command Qualifier Is Now Obsolete	D-7
D.1.4.6	Cyclic Redundancy Checking Emulation Improvements	D-8
D.1.4.7	Pressing Ctrl/T to Obtain Information About BACKUP Operations	D-8

E VMS Version 5.1 Features

E.1	VMS Version 5.1 Support for Compound Documents	E-1
E.1.1	VMS Commands and Utilities	E-1
E.1.1.1	Displaying RMS File Tags	E-2
E.1.1.1.1	DIRECTORY/FULL	E-2
E.1.1.1.2	ANALYZE/RMS_FILE	E-3
E.1.1.2	Creating RMS File Tags	E-3
E.1.1.3	Preserving RMS File Tags and DDIF Semantics	E-4
E.1.1.3.1	COPY Command	E-4
E.1.1.3.2	VMS Mail Utility	E-5
E.1.1.4	APPEND Command	E-5
E.1.2	DDIF Support in a Heterogeneous Environment	E-5
E.1.2.1	EXCHANGE/NETWORK Command	E-5
E.1.2.2	COPY Command	E-6
E.1.2.3	VMS Mail Utility	E-6
E.1.2.4	DDIF File Access Within a Mixed-Version Cluster	E-7
E.1.3	VMS RMS Interface Changes	E-7
E.1.3.1	Programming Interface for File Tagging	E-7
E.1.3.2	Accessing a Tagged File	E-9
E.1.3.2.1	File Accesses That Do Not Sense Tags	E-10
E.1.3.2.2	File Accesses That Sense Tags	E-10
E.1.3.3	Preserving Tags	E-12
E.1.4	Distributed File System Support for DDIF Tagged Files	E-12
E.1.5	VMS RMS Errors	E-13
E.2	EXCHANGE/NETWORK Command	E-13

Index

Examples

8-1	Showing Device Characteristics Using the SDA SHOW DEVICE Command	8-2
19-1	Using the CASE_SENSITIVE= Option	19-2
B-1	Sample AUTOGEN Command Procedure	B-53
B-2	Sample Transaction Log File	B-74
E-1	Tagging a File	E-8
E-2	Accessing a Tagged File	E-11

Figures

5-1	VMS Version 5.0 Queue Manager	5-2
5-2	VMS Version 5.5 Queue Manager	5-3
9-1	A LAT Network Configuration	9-12
16-1	IOSB Structure	16-2
17-1	Example of SETMODE Item List	17-2
A-1	System Based on XMI/VMEbus	A-9
A-2	Little-Endian Versus Big-Endian Byte Alignment	A-11
A-3	VMEbus DMA to and from VAX Host	A-12
A-4	VMEbus Map Register	A-13
A-5	Mapping of Programmed I/O Access from User Space	A-14
A-6	VME Map Register Descriptor (VME_MD)	A-23
A-7	FDDI Frame Format	A-56
A-8	Ethernet Frame Format	A-57
A-9	Ethernet Frame Format with PAD Option	A-57
A-10	IEEE 802.3 Frame Format	A-58
A-11	FDDI Frame with Mapped Ethernet Packet Format	A-58
A-12	FDDI Frame with Mapped Ethernet with PAD Option Packet Format	A-59
A-13	FDDI Frame with 802 Packet Format	A-59
A-14	CSMA/CD Frame with 802 Packet Format	A-59
A-15	FDDI Frame with 802E Packet Format	A-60
A-16	CSMA/CD Frames, 802E, 802.1 SNAP, and 802.1 PID Packet Format	A-60
B-1	VAX 6000-400 Series Vector-Present Processor Configuration	B-6
B-2	VAX 9000 Series Vector-Present Processor Configuration	B-7
B-3	Life of a Vector Consumer	B-9
B-4	Sample Debit/Credit Transaction Execution	B-35
B-5	Participants in a Distributed Transaction Example	B-38
B-6	DECwindows Screen Number Dialog Box	B-45
B-7	DECwindows Customize Screen Number Dialog Box	B-46
B-8	SCSI_NOAUTO System Parameter	B-56
B-9	Sample Transaction Log File Configuration on Two-Node VAXcluster	B-71
B-10	TRANSACTION Class Record Format	B-99
B-11	VECTOR Class Record Format	B-104
C-1	DNS Namespace	C-6
C-2	Valid Character Codes for DNS Simple Names	C-8
C-3	Additional Character Codes Allowed in Quoted Simple Names	C-9
C-4	Partitioned Namespace	C-14
C-5	Namespace with Replicated Directories	C-15

Tables

1-1	Summary of VMS Version 5.5 Software Features	1-1
2-1	Other VMS Version 5.5 DCL Commands and Qualifiers	2-1
2-2	F\$MESSAGE Keywords	2-7
3-1	F\$GETQUI Items	3-8
13-1	Fast-Vector Math Routines	13-1
13-2	Input Argument Restrictions	13-2
16-1	DECdtm System Services Changes	16-1
16-2	SDA Utility Changes	16-2
17-1	LAT\$C_ENT_NODE Setmode Item Codes	17-3
17-2	LAT\$C_ENT_SERVICE Setmode Item Codes	17-5
17-3	LAT\$C_ENT_LINK Setmode Item Codes	17-6
17-4	LAT\$C_ENT_PORT Setmode Item Codes	17-6
17-5	LAT\$C_ENT_NODE Sensemode Item Codes	17-8
17-6	Node Service Subblock Item Codes	17-10
17-7	Node Counters Item Codes	17-11
17-8	Protocol Error Item Codes	17-13
17-9	LAT\$C_ENT_SERVICE Sensemode Item Codes	17-13
17-10	Service Node Subblock Item Codes	17-14
17-11	Service Counters Subblock Item Codes	17-14
17-12	LAT\$C_ENT_LINK Sensemode Item Codes	17-15
17-13	Link Counters Item Codes	17-16
17-14	LAT\$C_ENT_PORT Sensemode Item Codes	17-16
22-1	FIB Fields (Movefile)	22-1
A-1	Summary of VMS Version 5.4-3 Software Features	A-1
A-2	Mapped Defaults for XMI and VME Interrupt Request Levels	A-12
A-3	Driver Entry Point Routines	A-15
A-4	Driver Notions Porting from UNIX to VMS	A-16
A-5	DR11-W VME Driver Code Contents	A-35
A-6	Required Size for P5 Diagnostics Buffer on FDDI Devices	A-54
B-1	Summary of VMS Version 5.4 Software Features	B-1
B-2	Settings of VECTOR_PROC System Parameter	B-11
B-3	System Messages Relating to Vector Processing	B-17
B-4	Summary of Exception Conditions	B-29
B-5	Summary of New and Enhanced DCL Commands	B-41
B-6	Arguments to the /ALGORITHM Qualifier	B-66
B-7	Descriptions of TRANSACTION Class Record Fields	B-100
B-8	Descriptions of VECTOR Class Record Fields	B-104
B-9	Descriptions of Additions to System Record Fields	B-104
C-1	DNS Item-Code Arguments	C-41
D-1	UAF Process Quotas for the BACKUP Account	D-6
D-2	Suggested Values for UAF Process Quotas	D-7
E-1	Tag Support Item Codes	E-7

Preface

Intended Audience

This book is intended for general users, system managers, and programmers who use the VMS operating system.

Document Structure

This manual is organized as follows:

- Part I, Summary of New Features, contains a summary of the new VMS Version 5.5 software features.

Note

It is important that you read Part I first for a complete overview of the VMS Version 5.5 new features.

- Part II, General User Features, describes new features primarily of interest to general users of the VMS operating system. The chapters within provide information about new DCL commands and qualifiers that have general applications, changes introduced by the new batch and print queuing system and new VMS system messages.
- Part III, System Management Features, describes new features that are applicable to the tasks performed by system managers. These features relate specifically to the following VMS components:
 - Batch and Print Queuing System
 - LADCP
 - Cluserwide Tape Sharing
 - VMS Volume Shadowing Phase II
 - LAT
 - License Management Facility
 - Movefile Operations
- Part IV, Programming Features, describes new features that support programming tasks. The chapters within provide information about the following components of the VMS operating system:
 - System Services Support for the Batch and Print Queuing System
 - Run-Time Library Routines
 - VMS Debugger

- DECthreads
- VMS Data Transaction Processing (DECdtm)
- LAT I/O Functions (LAT \$QIO)
- Asynchronous Printer Support
- Support for Case Sensitivity
- System Dump Analyzer Utility
- Mailbox Driver Interface
- QIO-ACP Support for Moving Disk Files

This document includes five appendixes. The appendixes describe features that were new to previous VMS versions but are not yet documented in other printed manuals.

Associated Documents

Refer to the following documents for more detailed information about the VMS Version 5.5 software features described in this manual. For more information about these documents, see the *Overview of VMS Documentation* or contact your Digital representative.

- *VAX RMS Journaling Manual*
- *VAX Text Processing Utility Manual*
- *VMS Developer's Guide to VMSINSTAL*
- *VMS Accounting Utility Manual*
- *VMS Authorize Utility Manual*
- *VMS Backup Utility Manual*
- *VMS DCL Dictionary*
- *VMS DCL Concepts Manual*
- *VMS Debugger Manual*
- *VMS DECwindows User's Guide*
- *VMS Delta/XDelta Utility Manual*
- *VMS Device Support Manual*
- *VMS Device Support Reference Manual*
- *Overview of VMS Documentation*
- *VMS File Definition Language Facility Manual*
- *Guide to VMS Files and Devices*
- *VMS I/O User's Reference Manual: Part I*
- *VMS I/O User's Reference Manual: Part II*
- *VMS LAD Control Program (LADCP) Manual*
- *VMS LAT Control Program (LATCP) Manual*
- *VMS Librarian Utility Manual*
- *VMS RTL Library (LIB\$) Manual*

- *VAX MACRO and Instruction Set Reference Manual*
- *VMS System Messages and Recovery Procedures Reference Manual*
- *VMS Monitor Utility Manual*
- *VMS RTL Mathematics (MTH\$) Manual*
- *Introduction to VMS System Routines*
- *VMS Record Management Services Manual*
- *VMS System Dump Analyzer Utility Manual*
- *Guide to VMS System Security*
- *VMS System Generation Utility Manual*
- *Introduction to VMS System Management*
- *Guide to Maintaining a VMS System*
- *Guide to Setting Up a VMS System*
- *VMS SYSMAN Utility Manual*
- *Introduction to VMS System Services*
- *VMS System Services Reference Manual*
- *VMS User's Manual*
- *VMS Utility Routines Manual*
- *VMS VAXcluster Manual*
- *VAX Volume Shadowing Manual*
- *VMS Volume Shadowing Manual*
- *X and Motif Quick Reference Guide*
- *VMS Version 5.5 Upgrade and Installation Manual*
- *VMS Version 5.5 Release Notes*

Conventions

The following conventions are used in this manual:

Ctrl/x

A sequence such as Ctrl/x indicates that you must hold down the key labeled Ctrl while you press another key or a pointing device button.

PF1 x

A sequence such as PF1 x indicates that you must first press and release the key labeled PF1, then press and release another key or a pointing device button.

Return

In examples, a key name is shown enclosed in a box to indicate that you press a key on the keyboard. (In text, a key name is not enclosed in a box.)

...

In examples, a horizontal ellipsis indicates one of the following possibilities:

- Additional optional arguments in a statement have been omitted.
- The preceding item or items can be repeated one or more times.
- Additional parameters, values, or other information can be entered.

A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.

()

In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.

[]

In format descriptions, brackets indicate that whatever is enclosed within the brackets is optional; you can select none, one, or all of the choices. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)

{ }

In format descriptions, braces surround a required choice of options; you must choose one of the options listed.

red ink

Red ink indicates information that you must enter from the keyboard or a screen object that you must choose or click on.

For online versions of the book, user input is shown in **bold**.

boldface text

Boldface text represents the introduction of a new term or the name of an argument, an attribute, or a reason.

Boldface text is also used to show user input in online versions of the book.

italic text

Italic text represents information that can vary in system messages (for example, Internal error *number*).

UPPERCASE TEXT

Uppercase letters indicate that you must enter a command (for example, enter OPEN/READ), or they indicate the name of a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.

-

Hyphens in coding examples indicate that additional arguments to the request are provided on the line that follows.

numbers

Unless otherwise noted, all numbers in the text are assumed to be decimal. Nondecimal radices—binary, octal, or hexadecimal—are explicitly indicated.

Part I

Summary of New Features

This part contains a summary of the new features supported by Version 5.5 of the VMS operating system.

Part I

Summary of New Features

The following features are new to this version of the software. For a complete list of changes, see the release notes.

Summary of New VMS Version 5.5 Features

This chapter provides a summary of the new software features supported by VMS Version 5.5 and a brief overview of new books. For information about new and enhanced hardware, see the *VMS Version 5.5 Release Notes*.

1.1 New Software Features

Table 1-1 provides a summary of new features supported by VMS Version 5.5.

Table 1-1 Summary of VMS Version 5.5 Software Features

VMS Version 5.5 General User Features	
DCL Commands and Lexical Functions	<p>New and enhanced DCL commands that provide the following capabilities:</p> <ul style="list-style-type: none"> Specify the size of DECram virtual disks Optionally force an improperly dismounted disk volume to be rebuilt to obtain the correct free block count Maintain the integrity of TZK10 tape cartridges by retensioning the tape during rewinding or unloading Create a file to log messages during CDA conversion operations
Lexical Functions	<p>The F\$MESSAGE lexical function has been modified to let you specify the system message component for which information is to be returned.</p> <p>For the new batch and print queuing system, the F\$ENVIRONMENT lexical function has a new item, VERIFY_PREFIX, which returns the prefix control string for verified command lines as part of the enhanced VMS Batch and Print Queuing System.</p> <p>For the new batch and print queuing system, the F\$GETQI lexical function returns information about the AUTOSTART feature and about user-specified job retention.</p>

(continued on next page)

Summary of New VMS Version 5.5 Features

1.1 New Software Features

Table 1-1 (Cont.) Summary of VMS Version 5.5 Software Features

VMS Version 5.5 General User Features	
Batch and Print Queuing System	<p>The batch and print queuing system provides the following improvements:</p> <ul style="list-style-type: none"> • Improved reliability and availability of batch and print queues • Improved performance in large configurations • Greater emphasis on clusterwide operations
System Messages	<p>New, updated, or previously undocumented system messages are included for a number of VMS facilities. The messages chapter also incorporates messages that were published in the <i>VMS Version 5.4 Release Notes</i>.</p>
VMS Version 5.5 System Management Features	
Batch and Print Queuing System	<p>Version 5.5 supports clusterwide queue management, a new queue database, and an autostart feature that simplifies queue startup and ensures high availability of queues.</p>
LADCP Utility	<p>Allows you to configure and control the local area disk (LAD) protocol on VMS host systems.</p>
Clusterwide Tape Serving	<p>Through the implementation of a tape mass storage control protocol (TMSCP), allows users on a node in a cluster to access magnetic tape devices physically connected to any other node in the cluster.</p>
Volume Shadowing Phase II	<p>Phase II supports a new SYSGEN parameter, SHADOW_MBR_TMO, that lets you specify the timeout period for recovering a shadow set member before it is removed from a shadow set. Phase II also provides support for SCSI (Small Computer System Interface) devices.</p>
LAT	<p>You can now use the SET HOST/LAT command to establish outbound (forward) LAT connections. New LATCP commands permit you to display information about various LAT entities, and there is a new startup procedure for LAT software.</p>
License Management Facility	<p>LMF has been enhanced to allow the transfer of licenses between databases and the registration of a license in another license database. System managers or privileged users can now attach a list of names to product licenses and software vendors can issue PAKs with the RESERVE_UNITS option. Another enhancement allows license managers to perform operations on groups of licenses.</p>

(continued on next page)

Summary of New VMS Version 5.5 Features

1.1 New Software Features

Table 1-1 (Cont.) Summary of VMS Version 5.5 Software Features

VMS Version 5.5 General User Features	
DCL Support for Movefile Operations	Three DCL commands, SET FILE, DIRECTORY/FULL and DUMP/HEADER, have been enhanced to support movefile operations that permit you to move the contents of a file, or part of the contents of a file, to a new disk location.
VMS Version 5.5 Programming Features	
System Services Support for New Features	Various system services have been modified to support the batch and print queuing system, to provide additional support for the LIB\$GETQUI run-time library routine and to support new DECdtm features.
RTL Routines	<p>The <i>fast-vector</i> math routines provide alternative math functions that offer significantly higher performance.</p> <p>LIB\$GETQUI has been enhanced to support the new batch and print queuing system.</p> <p>The PPL\$ run-time library provides enhanced unique naming functionality and spin/wait options for several blocking synchronization routines.</p>
VMS Debugger	Provides enhanced support for programs that have multiple threads of execution within a VMS process, including any program that uses DECthreads or POSIX 1003.4a services.
VMS DECthreads	This version of VMS supports Digital's Multithreading Run-Time Library, a library of portable routines used for creating and controlling multiple threads of execution within the address space provided by a single process.
DECdtm	Version 1.1 of the DECdtm services provides reason codes on transaction abort and transaction timeouts.
LAT \$QIO	The LAT function SET MODE provides the capability for creating and deleting LAT entities such as nodes, services, ports, and links, and to modify parameters of those LAT entities.
I/O Drivers	<p>The terminal driver interface supports connection of an asynchronous printer to a terminal port using modem signals for flow control.</p> <p>The mailbox driver now waits until a channel with the requested access direction is assigned to the mailbox.</p>
Case-Sensitive Language Support	The linker and MACRO now support case-sensitive programming languages. The linker preserves the mixture of upper and lowercase characters used in character-string arguments and MACRO now enables programmers to specify the case sensitivity of global symbol definitions.
System Dump Analyzer (SDA) Utility	The utility has been modified to provide support for transaction processing and a new symbol, TMSCP, for the tape mass storage control protocol server.

(continued on next page)

Summary of New VMS Version 5.5 Features

1.1 New Software Features

Table 1-1 (Cont.) Summary of VMS Version 5.5 Software Features

VMS Version 5.5 Programming Features	
DOCUMENT/CONVERT Command	You can now specify a single message file for messages generated by the input and output converters during the CDA conversion process. Digital CDA Base Services components, other than the command line interface to view and convert documents, are installed with DECwindows Motif Version 1.0.
QIO Support for Moving Disk Files	The movefile feature permits you to move all or part of the contents of a file to a new disk location. Typically, this might be used as part of a disk defragmentation application.

1.2 Announcing the New VMS Dependability Handbook

The VMS Version 5.5 documentation set includes a new handbook entitled *Building Dependable Systems: The VMS Approach*. A dependable computing system is one that can be counted on to always provide services to its users when those services are needed. The new handbook addresses the building blocks that make up a dependable system and explains basic dependability principles. It also provides practical techniques for utilizing the dependability features of VAX systems with those of the VMS operating system and layered software products to help you form a dependable computing system. *Building Dependable Systems: The VMS Approach* is included with the VMS Version 5.5 Base Documentation Set; it can also be ordered separately. See the *Overview of VMS Documentation* for ordering information.

1.3 Announcing the New BACKUP Utility Guide

A new manual, *Using VMS BACKUP*, is available to help users complete common tasks with the VMS Backup Utility (BACKUP). Intended as a companion to the *VMS Backup Utility Manual*, *Using VMS BACKUP* includes information about disk and tape operations; backing up and restoring files, directories, and disks; troubleshooting; and creating your own BACKUP command procedures.

Using VMS BACKUP is available on your VMS system disk (SYS\$EXAMPLES:USING_BACKUP.*) in DECW\$BOOK, LINE, and PS format.

Part II

General User Features

This part contains the following chapters:

- Chapter 2, DCL Commands and Lexical Functions
- Chapter 3, Batch and Print Queuing System
- Chapter 4, VMS System Messages

Part II

General User Features

- General system information
- General system settings
- General system status
- General system logs

DCL Commands and Lexical Functions

This chapter includes information about new qualifiers for various DCL commands and about a new lexical function:

- The /SIZE qualifier for the INITIALIZE command that supports DECram disks.
- The /REBUILD[=FORCE] qualifier for the SET VOLUME command that forces the building of a new disk volume, thereby updating the free block count in the disk volume's lock value block.
- The /RETENSION qualifier for the SET MAGTAPE command that defines the default characteristics associated with a specific magnetic tape device for subsequent file operations.
- The /MESSAGE_FILE qualifier for the CONVERT/DOCUMENT command. The qualifier creates a message file to which messages are logged during the conversion of your document. To use this qualifier, you must install the DEC CDA Base Services shipping with VMS DECwindows Motif Version 1.0 or later.
- An enhancement to the F\$MESSAGE lexical function that permits you to specify the system message component for which information is to be returned.

Table 2-1 lists other DCL commands and qualifiers that support specific Version 5.5 new features described in other chapters of this manual.

Table 2-1 Other VMS Version 5.5 DCL Commands and Qualifiers

DCL Command/Qualifier	Location
DISABLE AUTOSTART	Chapter 5
ENABLE AUTOSTART	Chapter 5
INITIALIZE/QUEUE/AUTOSTART_ON=(node-list)	Chapter 5
LICENSE COPY	Chapter 10
LICENSE ISSUE/PROCEDURE	Chapter 10
LICENSE MODIFY/RESERVE	Chapter 10
LICENSE MOVE	Chapter 10
LICENSE subcommand/ALL	Chapter 10
MACRO/NAMES	Chapter 19
PRINT/RETAIN	Chapter 3
SET ENTRY/RETAIN	Chapter 3

(continued on next page)

Table 2-1 (Cont.) Other VMS Version 5.5 DCL Commands and Qualifiers

DCL Command/Qualifier	Location
SET FILE/NOMOVE	Chapter 11
SET HOST/LAT	Chapter 9
SET PREFIX	Chapter 3
SET TERMINAL/COMMSYNC	Chapter 18
SHOW ENTRY	Chapter 3
SHOW QUEUE	Chapter 3
START/QUEUE/AUTOSTART_ON=(node-list)	Chapter 5
START/QUEUE/MANAGER	Chapter 5
STOP/QUEUE/MANAGER/CLUSTER	Chapter 5
STOP/QUEUE/NEXT	Chapter 5
STOP/QUEUE/RESET	Chapter 5
STOP/QUEUES/ON_NODE	Chapter 5
SUBMIT/NOTE	Chapter 3
SUBMIT/RETAIN	Chapter 3

2.1 DCL Command Enhancements

This section describes the enhanced DCL commands supported by VMS Version 5.5.

INITIALIZE/SIZE

The INITIALIZE/SIZE=*n* command specifies the size of the DECram virtual disk to be allocated from available memory. DECram is a layered product that is used to create virtual disks in system memory. See DECram documentation for more information about DECram.

Format

INITIALIZE/SIZE=*n* device-name[:] volume-label

Description

The INITIALIZE command now accepts the /SIZE=*n* qualifier in support of DECram virtual disks (device type DT\$_RAM_DISK). The /SIZE=*n* qualifier specifies the size of the virtual disk to be allocated from available memory. This allows you to define the size of the DECram device at initialization time. Note that *n* cannot exceed 524,280 blocks. A DECram virtual disk requires one page of system space per block of virtual disk space allocated.

To deallocate space for a DECram virtual disk, specify /SIZE=0 with the INITIALIZE command. All resources specifically allocated to the DECram virtual disk will be returned to the system.

See the *VMS DCL Dictionary* for more information about the INITIALIZE command.

SET VOLUME/REBUILD[=FORCE]

Forces a disk volume to be rebuilt, causing the free block count value to be updated.

Requires write (W) access to the index file on the volume. If you are not the owner of the volume, requires either a system user identification code (UIC) or SYSPRV (system privilege) privilege.

Format

SET VOLUME/REBUILD[=[NO]FORCE] device-name[:],...

Description

The SET VOLUME/REBUILD command is used to recover the caching that was in effect at the time when a disk volume was dismounted improperly (such as during a system failure or a cluster transition). The FORCE option forces the disk volume to be rebuilt unconditionally, which updates the free block count in the disk volume's lock value block. The default is NOFORCE.

During a cluster transition, the free block count that is maintained on a lost primary node might not be made available to the new primary node. As a result, the free block count on the new primary node might be incorrect. Because of this free block count discrepancy, the number of free blocks available for use on a disk might be higher or lower than the actual free block count. Attempts to use the free blocks might result in allocation failures.

The SET VOLUME/REBUILD=FORCE command should be issued as soon as the free blocks discrepancy is discovered, especially if a new primary node is identified for a mounted disk volume following a cluster state transition.

See the *VMS DCL Dictionary* for more information about the SET VOLUME command.

SET MAGTAPE/RETENSION

Defines the default characteristics associated with a specific magnetic tape device for subsequent file operations. The /RETENSION qualifier moves a TZK10 tape cartridge to the end of the tape and then back to the beginning of the tape.

Format

SET MAGTAPE/RETENSION device-name[:]

Description

The SET MAGTAPE command uses the /RETENSION qualifier to move a TZK10 tape cartridge to the end of the tape and then back to the beginning of the tape. Using the /RETENSION qualifier on a regular basis helps maintain the integrity of TZK10 tape cartridges.

You must use the /RETENSION qualifier with either the /REWIND or /UNLOAD qualifier. The /RETENSION qualifier completes its action before /REWIND or /UNLOAD. Use /RETENSION/REWIND when you want the tape cartridge to remain loaded in the drive. Use /RETENSION/UNLOAD when you want to unload the tape cartridge after the retention operation.

This qualifier affects TZK10 tape cartridge drives only, and causes the following error message on other SCSI tape cartridge drives:

%SET-I-FUNCNOTSUP, 'device-name' does not support /RETENSION; qualifier ignored.

The /RETENSION qualifier has no effect on non-SCSI tape drives.

See the *VMS DCL Dictionary* for more information about the SET MAGTAPE command.

CONVERT/DOCUMENT/MESSAGE_FILE=filespec

Allows you to specify a file for logging messages during conversion.

Note

The DEC CDA Base Services for VMS DECwindows Motif Version 1.0 or later must be installed in order to use the /MESSAGE_FILE qualifier and new versions of the DEC CDA Base Services converters.

Format

CONVERT/DOCUMENT/MESSAGE_FILE= input-filespec output-filespec

Description

The CONVERT/DOCUMENT command converts documents from one format to another for the purpose of sharing information among different applications. The default input and output file format is DDIF (Digital Document Interchange Format), a standard format for the storage and interchange of compound documents, which can include text, graphics, and images.

The /MESSAGE_FILE qualifier creates a file to which informational and error messages are logged during the conversion.

Example

```
$ CONVERT/DOCUMENT/OPTIONS=MY_OPTIONS.CDA$OPTIONS -  
_$ MY_INPUT.DTIF/FORMAT=DTIF MY_OUTPUT.DDIF/FORMAT=DDIF  
_$ /MESSAGE_FILE=MY_MSGS.MSG
```

This command converts an input file named MY_INPUT.DTIF, which has the DTIF format, to an output file named MY_OUTPUT.DDIF, which has the DDIF format. The specified options file is named MY_OPTIONS.CDA\$OPTIONS, and the message file is named MY_MSGS.MSG.

2.2 F\$MESSAGE Lexical Function

The **message-component-list** argument for the F\$MESSAGE lexical function allows you to specify the system message component for which information is to be returned.

F\$MESSAGE(status-code [,message-component-list])

Argument

message-component-list

The system message component or components to be returned. If this parameter is null or unspecified, then all system message components are returned.

Table 2-2 describes the valid system message component keywords.

Table 2-2 F\$MESSAGE Keywords

Component Keyword	Information Returned
FACILITY	Facility name
SEVERITY	Severity level indicator
IDENT	Abbreviation of message text
TEXT	Explanation of message

Note that when the FACILITY, SEVERITY, and IDENT code keywords are specified (individually or in combination), the resulting message code is preceded by the percent sign (%) character. The individual parts of the message code are separated by hyphens when multiple code keywords are specified.

When only the TEXT keyword is specified, the resulting text is not preceded by any character. When the TEXT keyword is specified with the FACILITY, SEVERITY, or IDENT code keyword, the message code is separated from the text by a comma and a space (,).

Examples

```
1. $ ERROR_INFO = F$MESSAGE(%X1C,"TEXT")
   $ SHOW SYMBOL ERROR_INFO
   ERROR_INFO = "EXCEEDED QUOTA"
```

This example shows the system message component that is returned by using the keyword TEXT.

DCL Commands and Lexical Functions

2.2 F\$MESSAGE Lexical Function

```
2. $ SUBMIT IMPORTANT.COM
   $ SYNCHRONIZE /entry='$ENTRY'
   $ IF $STATUS THEN EXIT
   $!
   $ JOB_STATUS = $STATUS
   $!
   $ IF "%JOBDELETE" .EQS. F$MESSAGE (JOB_STATUS, "IDENT")
   $ THEN
       .
       .
       .
   $ ELSE
       $ IF "%JOBABORT" .EQS. F$MESSAGE (JOB_STATUS, "IDENT")
       $ THEN
           .
           .
           .
       $ ELSE
           .
           .
           .
       $ ENDIF
   $ ENDIF
   .
   .
   .
```

This command procedure submits a batch job and waits for it to complete. Upon successful completion, the procedure exits. If the job completes unsuccessfully, more processing is done based on the termination status of the batch job.

The first command submits the command procedure IMPORTANT.COM. The second command, SYNCHRONIZE, tells the procedure to wait for the job to finish. The third command determines if the job completed successfully and, if so, the procedure exits. The next command saves the status in a symbol.

The first IF statement uses F\$MESSAGE to determine whether the job was deleted before execution. If so, it does some processing, possibly to resubmit the job or to inform a user via MAIL.

The next IF statement uses F\$MESSAGE to determine whether the job was deleted during execution. As a result, some cleanup or human intervention may be required, which would be done in the THEN block.

If neither IF statement was true, then some other unsuccessful status was returned. Other processing, which would be done in the block following the ELSE statement, might be required.

Batch and Print Queuing System

This chapter contains information about the new batch and print queuing system that is of interest to general users.

3.1 Changes to SHOW ENTRY

This section describes changes to the SHOW ENTRY display and command.

3.1.1 Change in Format of SHOW ENTRY Display

In the previous batch and print queuing system, the SHOW ENTRY command returned a display similar to the following:

Jobname	Username	Entry	Blocks	Status
MYJOB	HERSHEY	6		Retained on completion
On generic batch queue CLUSTER_BATCH				

In the new batch and print queuing system, the format for the SHOW ENTRY display is changed and appears similar to the following:

Entry	Jobname	Username	Blocks	Status
6	MYJOB	HERSHEY		Retained on completion
On stopped generic batch queue CLUSTER_BATCH				
Completed 28-MAR-1991 17:52 on queue NODE_BATCH				

The new display makes it easier for a user to locate a job's entry number. This is important because the entry number is needed for the SET ENTRY and DELETE /ENTRY commands. The new display also includes the state of the queue in which the job is currently located.

3.1.2 SHOW ENTRY Command Accepts Job Names

In VMS Version 5.0, the SHOW ENTRY command was added to let users display information about their batch and print jobs without having to view other queue information. The SHOW ENTRY command accepted any of the following values for its parameter:

- No value, to display all of a user's jobs
- A single entry number or a list of entry numbers, to display only those jobs specified
- The \$ENTRY symbol, to display the job most recently added by that process (this feature was added with VMS Version 5.2)

Batch and Print Queuing System

3.1 Changes to SHOW ENTRY

In the new batch and print queuing system, the SHOW ENTRY command also accepts a job name as a legal value for its parameter. The command SHOW ENTRY *job_name* displays all of the user's jobs having the specified job name, as shown in the following example:

```
$ SHOW ENTRY CHECKNODE
```

Entry	Jobname	Username	Blocks	Status
38	CHECKNODE	HERSHEY		Holding
	On stopped batch queue NODEA_BATCH			
167	CHECKNODE	HERSHEY	2	Pending
	On stopped printer queue NODEB_PRINT			
605	CHECKNODE	HERSHEY		Pending
	On stopped batch queue NODEC_BATCH			

Wildcards are allowed. You can also specify lists using any combination of valid parameters. For example, the following command displays entry 605 and all entries with job names starting with "W":

```
$ SHOW ENTRY 605, W*
```

By specifying a job name with the SHOW ENTRY command, users can view information about their entries without having to remember the entry numbers assigned to the jobs. This is helpful for users with many jobs in the system.

For more information about the SHOW ENTRY command, see the *VMS DCL Dictionary*.

3.1.3 New Stalled Job State

Previously, when a queue physically stalled, the SHOW ENTRY command output for the executing job would display the status of the job as "Executing" even though the job was stalled. For example, if queue NODEA_PRINT were stalled, a SHOW ENTRY command would display the following:

Jobname	Username	Entry	Blocks	Status
MYJOB	HERSHEY	6	238	Executing
On printer queue NODEA_PRINT				

Thus, the user might incorrectly believe that the job was processing.

In the new batch and print queuing system, when the physical device to which a queue is assigned is stalled, the job's status now appears as "Stalled", as shown in the following example:

Entry	Jobname	Username	Blocks	Status
6	MYJOB	HERSHEY	238	Stalled
On stalled printer queue NODEA_PRINT				

The new display also includes the state of the queue in which the job is located.

3.2 Change in Format of SHOW QUEUE Display

In the previous batch and print queuing system, the SHOW QUEUE command returned a display similar to the following:

Batch and Print Queuing System

3.2 Change in Format of SHOW QUEUE Display

Batch queue NODE_BATCH, on NODE22::

Jobname	Username	Entry	Status
-----	-----	-----	-----
SET	HERSHEY	6	Executing

In the new batch and print queuing system, the display for the SHOW QUEUE command is changed to appear similar to the following:

Batch queue NODE_BATCH, busy, on NODE22::

Entry	Jobname	Username	Status
-----	-----	-----	-----
6	SET	HERSHEY	Executing

The new display makes it easier for you to locate a job's entry number.

3.3 User-Specified Job Retention

In the previous batch and print queuing system, system managers could use the /RETAIN qualifier with the INITIALIZE/QUEUE, START/QUEUE, or SET QUEUE command to establish job retention policy for a particular queue.

In the new batch and print queuing system, users can also use the /RETAIN qualifier with the PRINT, SUBMIT, or SET ENTRY command to specify the circumstances under which they want their jobs to be retained in a queue.

3.3.1 Uses for User-Specified Job Retention

Specifying job retention can be useful for the following reasons:

- Changes to the SHOW ENTRY and SHOW QUEUE displays include the date and time at which a retained job completed and the queue on which it executed. This information can help you determine which printer a print job's output was sent.
- As with previous versions, the SHOW ENTRY and SHOW QUEUE displays for jobs retained on error also include the unsuccessful status message. This message can help you determine why a job did not complete successfully.

Without job retention, no record of a job is left in a queue after a job completes. However, when a job is retained in the queue, you can issue the SHOW QUEUE command after the job completes to see the status of the job. For example:

```
$ SHOW QUEUE DOC$LN03
```

Server queue DOC\$LN03, stopped, on NEWTON:: mounted form DEFAULT

Entry	Jobname	Username	Blocks	Status
-----	-----	-----	-----	-----
436	DOCPLAN	HERSHEY	8	Retained on error
%JBC-F-JOBABORT, job aborted during execution				
Completed 4-APR-1991 20:15 on queue DOC\$LN03				

3.3.2 Job Retention Command Syntax

To specify that you want your job to be retained, use the /RETAIN qualifier with the PRINT, SUBMIT, or SET ENTRY commands as shown in the following syntax example:

```
PRINT/RETAIN=option filespec[,...]
```


Batch and Print Queuing System

3.3 User-Specified Job Retention

where *option* can be one of the following:

- **ALWAYS**—Holds the job in the queue regardless of the job's completion status.
- **DEFAULT**—Holds the job in the queue as specified by the queue's retention policy.
- **ERROR**—Holds the job in the queue only if the job completes unsuccessfully.
- **UNTIL=*time-value***—Holds the job in the queue for the specified length of time, regardless of the job's completion status.

Note

You cannot specify the /NORETAIN qualifier with the commands PRINT, SUBMIT, and SET ENTRY (as system managers can with the commands INITIALIZE/QUEUE, START/QUEUE, and SET QUEUE); however, you can specify /RETAIN=DEFAULT with those commands. The default option holds the job in the queue as specified by the queue's retention policy. If the system manager has not specified retention for the queue, the job is not retained.

3.3.3 How Job Retention Is Determined

Although you can now specify job retention options for your own jobs, the job retention option you specify may be overridden by the job retention option of the queue on which your job executed. If you submit or print a job to a generic queue, the generic queue's job retention setting may also override the job retention option you specify. This section describes how job retention is determined.

An execution queue's job retention setting takes precedence over a generic queue's job retention setting. However, if the job's completion status does not match the job retention setting (if any) on the execution queue, then the generic queue's job retention setting attempts to control job retention. If the job's completion status does not match the job retention setting (if any) on the generic queue, then the user-specified job retention setting is used. Jobs submitted directly to execution queues are not affected by job retention settings on generic queues.

If the execution queue's retention setting applies, the job is retained on the execution queue. Likewise, if the generic queue's retention setting applies, the job is retained on the generic queue. If the user-specified setting applies, the job is retained in the queue to which it was submitted.

The following example illustrates how the queue manager determines how and where to retain a job.

Suppose you submit a job to a generic queue and specify /RETAIN=ALWAYS, and the job completes successfully.

First, the queue manager compares the job's completion status to the execution queue's retention setting. If the queue is set with /RETAIN=ERROR (retains only jobs that complete unsuccessfully), the job is not retained in the execution queue because the error condition was not met.

The queue manager then compares the job's completion status to the generic queue's retention setting. If the generic queue has no retention setting, the queue manager's comparison again fails to retain the job.

Finally, the queue manager compares the job's completion status to the retention setting you specified for the job. This comparison reveals that the job should be retained. Because the user-specified setting leads the queue manager to retain the job, the job is held in the queue to which the job was submitted—in this case, the generic queue.

For more information about types of queues, see the INITIALIZE/QUEUE command in the *VMS DCL Dictionary*. For more information about setting retention options for queues, see the INITIALIZE/QUEUE, START/QUEUE, or SET QUEUE command in the *VMS DCL Dictionary*.

3.3.4 Timed Retention

Timed retention, which you specify using the UNTIL=*time-value* option, allows you to retain a job in the queue only as long as you need it. This eliminates the need to delete the job from the queue later.

For example, the following command retains the print job MYFILE in the queue until 7:31 on April 19, when the job will automatically be deleted from the queue.

```
$ PRINT/RETAIN=UNTIL=19-APR-1991:07:31:0.0 MYFILE.DAT
```

However, depending on the queue's job retention policy, the job might be retained indefinitely. The job retention policy set on the queue takes precedence over the user-specified job retention setting. Because system managers cannot specify timed job retention for a queue, any jobs retained as a result of a queue's setting are retained indefinitely.

If you specify the /RETAIN=UNTIL=*time-value* option, you must supply a time value. The time value is first interpreted as a delta time, then as a combination time, and finally as an absolute time. If you specify a delta time, the delta begins when the job completes. For example, if you specify PRINT /RETAIN=UNTIL="+3:00", the job will be retained for three hours after the job completes. For information about specifying time values, see the *VMS User's Manual*.

3.4 Batch Log Time-Stamps

Batch time-stamps are being introduced with the new batch and print queuing system. The ability to time-stamp your log files lets you use a full date and time prefix to identify batch runs and to verify that a batch job ran at the expected time.

This feature lets users set a prefix, commonly called a time-stamp, for verified DCL command lines. The enhancement uses the \$FAO (formatted ASCII output) system service to provide some flexibility in formatting the prefix. The FAO control string is limited to:

- Constants
- Special formatting directives (such as "!", "_", "^", "!", and "!n*c")
- Date/time directives ("!%D" and "!%T")
- Repeat counts ("!n(DD)")
- Output-field-length specifications ("!lengthDD")

Time-stamping occurs once for a verified command; continuation lines are padded with blanks. Image input and output lines are not prefixed or padded.

To use time-stamping, users must set a prefix control string with the new SET PREFIX command. Prefixing occurs any time that command verification is turned on with the SET VERIFY command or the F\$VERIFY lexical function. To determine the current prefix control string, use the F\$ENVIRONMENT lexical function with the new VERIFY_PREFIX item.

See Section 3.4.1 for more information about the new SET PREFIX command. See the *VMS DCL Dictionary* for more information about the SET VERIFY command or the F\$VERIFY lexical function. See Section 3.4.2 for more information about the new VERIFY_PREFIX item.

3.4.1 New DCL Command: SET PREFIX

The SET PREFIX command replaces the current verification prefix control string with a specified string. This allows you to prefix verified command lines with a custom string. This string is a limited FAO control string that specifies date and time information as well as information about constants and formatting controls (that is, tabs, form feeds, and so on). See the description of the F\$FAO lexical function in the *VMS DCL Dictionary* for more information about FAO control strings.

The first line of a verified command is prefixed with the result of the control string. Any continuation lines are prefixed with a blank string to make them flush with the first line of the command. Command input and output are not prefixed.

SET [NO]PREFIX string

Parameter

string

Specifies the new FAO control string to be used in generating a prefix to a verified command line. The following rules apply:

- No more than 64 characters are allowed in the control string.
- The resulting string can be no longer than 64 characters.
- Basic formatting FAO directives can be used ("!", "_", "^", "!", and "n*c").
- Time and date FAO directives can be used ("!%T" and "!!%D").
- Repeat counts can be used ("!n(DD)").
- Output-field-length specifications can be used ("!lengthDD").
- Combination of repeat count and output field length can be used ("!n(lengthDD)").

For more information about building an FAO control string, see the description of the F\$FAO lexical function in the *VMS DCL Dictionary*.

Example

```
$ SET VERIFY
$ @TEST
$ SET DEFAULT SYS$LOGIN
$ SHOW DEFAULT
USER$: [SMYTHE]
$ SET PREFIX "(!5%T) "
$ @TEST
(17:52) $ SET DEFAULT SYS$LOGIN
(17:52) $ SHOW DEFAULT
USER$: [SMYTHE]
```

This example demonstrates the difference between having no prefix for verification and having one. The first command turns on verification. (Verification must be on to see the prefix.) The second command invokes a test procedure to show what the output looks like without a prefix. The third and fourth lines reflect the contents of the test procedure invoked in the preceding command. The third command sets the prefix to an FAO control string so that the first five characters of the standard time will be shown for each command. The last command invokes the test procedure again to demonstrate what the output looks like with a prefix.

3.4.2 New Item for F\$ENVIRONMENT Lexical Function

A new item, VERIFY_PREFIX, has been added to the F\$ENVIRONMENT lexical function. The VERIFY_PREFIX item returns the prefix control string for verified command lines. Use the SET PREFIX command to set the control string. If procedure verification is in effect, then the control string will generate a prefix to which the verified line is appended. The return value for the VERIFY_PREFIX item is a character string.

Example

```
.
.
$ PROC_VER = F$ENVIRONMENT("VERIFY_PROCEDURE")
$ IMAGE_VER = F$ENVIRONMENT("VERIFY_IMAGE")
$ HOLD_PREFIX = F$ENVIRONMENT("VERIFY_PREFIX")
$ SET PREFIX "(!%T) "
$ SET VERIFY
.
.
$ TEMP = F$VERIFY(PROC_VER, IMAGE_VER)
$ SET PREFIX "'HOLD_PREFIX'"
```

This command procedure uses the F\$ENVIRONMENT lexical function to save the current procedure and image verification settings, as well as the current verification prefix string. The SET PREFIX command sets the verification prefix to be used in the current command procedure. It uses an FAO control string to produce the time each command is read by the command interpreter (DCL), surrounded by parentheses. Then the SET VERIFY command turns on both procedure and image verification. Subsequently, the F\$VERIFY lexical function is used to restore the original verification settings. The SET PREFIX command returns the verification prefix to its previous setting. Note how the symbol HOLD_PREFIX is used in the SET PREFIX command. This preserves casing and special characters in the stored string.

Batch and Print Queuing System

3.5 /NOTE Qualifier for SUBMIT Command

3.5 /NOTE Qualifier for SUBMIT Command

In the new batch and print queuing system, the SUBMIT command accepts the /NOTE qualifier. The /NOTE qualifier is used to specify a message string of up to 255 characters. This message string appears as part of the display on a SHOW QUEUE/FULL command and can thus be used to convey information concerning the job, as in the following example:

```
$ SUBMIT /HOLD MYJOB -
_$ /NOTE="ATTN OPERATOR: Mount tape ABCD before releasing job"
$ SHOW QUEUE/FULL SYS$BATCH
Generic batch queue SYS$BATCH
/GENERIC=(DEANNA_BATCH,TROI_BATCH,EMPATH_BATCH) /OWNER=[SYSTEM]
/PROTECTION=(S:E,O:D,G:R,W:W)

Entry  Jobname      Username      Status
-----
   38  MYJOB        HERSHEY      Holding
      Submitted 12-NOV-1991 17:56
      /NOTE="ATTN OPERATOR: Tape ABCD must be mounted before
      release of job" /PRIORITY=100
      File: _$1$DUA24:[HERSHEY]MYJOB.COM;2
```

The message specified with the /NOTE qualifier is also printed on the flag page of the log file and can be used to convey post-printing information, as in the following example:

```
$ SUBMIT /LOG_FILE /PRINTER=MYPRINTQUEUE MYJOB -
_$ /NOTE="Please send log file to second floor mailbox"
```

3.6 Changes to F\$GETQUI Lexical Function

In the new batch and print queuing system, the F\$GETQUI lexical function is enhanced to return information about the new AUTOSTART feature as it pertains to a queue. For more information about using F\$GETQUI, see the *VMS DCL Dictionary*. The AUTOSTART feature is described in detail in Section 5.4.

The F\$GETQUI lexical function is also enhanced to return information about user-specified job retention. See Section 3.3 for more information about user-specified job retention. In addition, the **object-id** argument to the DISPLAY_ENTRY function code now accepts a job name. For more information about the job name argument, see Section 3.1.2.

The **item** argument specifies the kind of information you want returned about a particular queue, job, file, form, or characteristic. Table 3-1 lists the new or enhanced item codes in the new batch and print queuing system.

Table 3-1 F\$GETQUI Items

Item	Return Type	Information Returned
AUTOSTART_ON	String	A list of node or node device pairs on which the autostart queue may be run. For information about autostart queues, see Section 5.4.

(continued on next page)

Batch and Print Queuing System

3.6 Changes to F\$GETQUI Lexical Function

Table 3-1 (Cont.) F\$GETQUI Items

Item	Return Type	Information Returned
QUEUE_AUTOSTART	String	FALSE. TRUE if the specified queue has been designated as an autostart queue. For information about autostart queues, see Section 5.4.
QUEUE_AUTOSTART_INACTIVE	String	FALSE. TRUE if the queue is an autostart queue that will not be automatically started. If TRUE, a START/QUEUE or INIT/QUEUE/START command must be issued to restart the queue. For information about autostart queues, see Section 5.4.
QUEUE_AVAILABLE	String	FALSE. TRUE if queue is processing work but is capable of processing additional work.
QUEUE_BUSY	String	FALSE. TRUE if queue cannot process additional jobs because of work in progress.
QUEUE_STOP_PENDING	String	FALSE. TRUE if queue will be stopped when work currently in progress has completed.
JOB_ERROR_RETENTION	String	FALSE. TRUE if the user requested that the specified job be held in the queue if it completes unsuccessfully.
JOB_RETENTION	String	FALSE. TRUE if the user requested that the specified job be held in the queue upon completion.
JOB_STALLED	String	FALSE. TRUE if the specified job is stalled because the physical device on which it was printing is stalled.
JOB_RETENTION_TIME	String	Shows the user-specified system time until which the specified job will be retained in the queue.
JOB_COMPLETION_TIME	String	Shows the time at which the execution of the specified job completed.
JOB_COMPLETION_QUEUE	String	Shows the name of the queue on which the specified job executed.

The **object-id** argument specifies either a name or a number of one or more objects about which F\$GETQUI is to return information.

In the new batch and print queuing system, the **object-id** argument is enhanced to accept a 1- to 39-character string when specified with the DISPLAY_ENTRY function. F\$GETQUI uses this string to restrict its search for a job or jobs. F\$GETQUI searches for job names that match the **object-id** input value for the given user name.

To direct F\$GETQUI to perform a wildcard search, specify the wildcard keyword as a flags argument. Wildcard characters (* and %) are allowed as part of the character-string value specified as the **object-id** argument.

Batch and Print Queuing System

3.6 Changes to F\$GETQUI Lexical Function

Example

```
$ this_node = f$edit(f$getsyi("scsnnode"),"collapse")
$ temp = f$getgui("cancel_operation")
$ set noon
$loop:
$ queue = f$getgui("display_queue","queue_name","*", "wildcard")
$ if queue .eqs. "" then goto endloop
$ if this_node .eqs. f$getgui("display_queue","scsnnode_name","*", "wildcard,freeze_context")
$ then
$   if .not. f$getgui("display_queue","queue_autostart","*", "wildcard,freeze_context")-
$     then start/queue 'queue'
$ endif
$ goto loop
$endloop:
$ set on
```

This command procedure looks at all queues associated with the local cluster node and starts any queue that is not marked as autostart.

The procedure starts by obtaining the node name of the local system and clearing the F\$GETQUI context. In addition, error handling is turned off for the subroutine so that if a queue had been started before, the resulting error from the START QUEUE command will not abort the command procedure.

Inside the subroutine, the F\$GETQUI function gets the next queue name in the queue list. If the result is empty, then it has reached the end of the list and exits the subroutine.

The next IF statement checks to see if the queue runs on the local node. If it does, the next statement checks if the queue is marked as an autostart queue. If the queue is marked as an autostart queue, it is started with the START command and the subroutine executes again.

The final command of the procedure restores DCL error handling to the previous setting.

VMS System Messages

This chapter alphabetically lists and describes system messages that have been added or revised since Version 5.4 of the *VMS System Messages and Recovery Procedures Reference Manual*. The following pages include new, updated, or previously undocumented system messages for the following VMS facilities:

- ANALDISK, Analyze/Disk_Structure Utility
- AUTHORIZE, Authorize Utility
- BACKUP, Backup Utility
- BUGCHECK, System Bugcheck
- CLI, Command Language Interpreter (DCL)
- CMA, DECthreads (Multithreading Run-Time Library)
- DISMOUNT, DISMOUNT Command
- DDTM, DECdtm Services
- INIT, INITIALIZE Command
- JBC, Job Controller
- LAVC, Local Area VAXcluster
- LAT, LAT Facility
- LMCP, Log Manager Control Program
- LOGIN, Login Processor
- MAIL, Mail Utility
- MOUNT, Mount Utility
- NCP, Network Control Program
- OPCOM, Operator Communication Process
- QMAN, Queue Manager
- SDA, System Dump Analyzer
- STDRV, System Startup Driver
- SYSBOOT, System Bootstrap Facility
- SYSGEN, System Generation Utility
- SYSTEM, VMS System Services

This chapter includes messages that were published in the *VMS Version 5.4 Release Notes*.

VMS System Messages

See Section B.4.2.1 for information on how to install and access an online help version of the *VMS System Messages and Recovery Procedures Reference Manual*.

ABORT, abort

Facility: SYSTEM, VMS System Services

Explanation: This message is returned under either of the following conditions:

- It is returned by \$START_TRANS if the DECdtm services are disabled on the local node or if the node does not have a transaction log.
- It is returned by \$END_TRANS if the transaction was aborted during processing.

User Action: On returns from \$START_TRANS, make sure the local node has a transaction log and the DECdtm transaction services are enabled. On returns from \$END_TRANS, check the abort reason code in the I/O status block to find out why the transaction aborted.

ABORTED, application aborted transaction via \$ABORT_TRANS service

Facility: DDTM, DECdtm Services

Explanation: The user program has aborted the transaction using the \$ABORT_TRANS service.

User Action: None.

ACCWAIT, waiting to access files in 'directory'

Facility: QMAN, Queue Manager

Explanation: When a node is booting, the queue manager can start up before the disk that contains queue and journal files is mounted. In such cases, all queuing-related requests wait and this message displays periodically to alert the system manager of the situation. An accompanying message describes the disk-related error.

User Action: Make sure the disk is mounted. Consult the accompanying message to diagnose any problems.

ACCWAITDONE, no longer waiting to access files in 'directory'

Facility: QMAN, Queue Manager

Explanation: When a node is booting, the queue manager can start up before the disk that contains queue and journal files is mounted. This message indicates that the queue manager is no longer waiting because the disk has been mounted or startup has completed.

User Action: None.

ACPINIT, LATACP initialized

Facility: LAT, LAT Facility

Explanation: The LATACP has been initialized.

User Action: None.

ACPNOCTL, insufficient resources - ACP CTL/P1 space limit reached

Facility: LAT, LAT Facility

Explanation: A request to add more information to the LATACP's database has been rejected because LATACP has insufficient resources to service the request.

User Action: Increase the value of SYSGEN parameter CTLPAGES or refer to the *VMS LAT Control Program (LATCP) Manual* for information on how to set a node limit to decrease LATACP's consumption of P1 space.

ACPNOKSTK, insufficient resources - ACP kernel stack limit reached

Facility: LAT, LAT Facility

Explanation: A request to add more information to the LATACP's database has been rejected because LATACP has insufficient resources to service the request.

User Action: Refer to the *VMS LAT Control Program (LATCP) Manual* for information on how to set a node limit to decrease LATACP's consumption of the kernel stack.

ACPNOVIRT, insufficient resources - ACP P0 space limit reached

Facility: LAT, LAT Facility

Explanation: A request to add more information to the LATACP's database has been rejected because LATACP has insufficient resources to service the request.

User Action: Refer to the *VMS LAT Control Program (LATCP) Manual* for information on how to set a node limit to decrease LATACP's consumption of P0 space, or determine the cause of the resource exhaustion and attempt to correct it by tuning the LATACP process.

ALERTED, thread execution has been canceled

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A thread has been requested to terminate by either the **cma_thread_alert** or **pthread_cancel** routine. DECthreads uses an alert to request that a thread terminate after first performing cleanup and shutdown operations.

User Action: If you do not want threads to terminate at the point where this alert is being delivered, you can use several routines (**cma_alert_disable_general**, **cma_alert_disable_asynch**, **cma_alert_restore**, **pthread_setcancel**, and **pthread_setasynccancel**) to specify points in the thread process where alerts cannot be delivered to the thread.

ALERT_NESTING, improper nesting of alert scope

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: An attempt was made to restore an inner scope after an enclosing outer scope had already been restored.

User Action: Examine the code to determine where the incorrect alert state variable was passed to the **cma_alert_restore** routine.

VMS System Messages

ALLOCMEM, error allocating virtual memory

Facility: JBC, Job Controller and QMAN, Queue Manager

Explanation: The job controller or queue manager encountered an error while allocating virtual memory. An accompanying message provides additional information.

User Action: Take action based on the accompanying message. You might need to run AUTOGEN to increase the SYSGEN parameter VIRTUALPAGECNT. If the accompanying message does not require you to keep the process dump, you can delete it.

ALRCURTID, a default transaction is currently defined

Facility: SYSTEM, VMS System Services

Explanation: An attempt was made to start a default transaction when the process already had a default transaction.

User Action: Either terminate the default transaction already in progress before starting a new one or start a new transaction as a nondefault transaction.

ARBTOOBIG, access rights block too big

Facility: SYSTEM, VMS System Services

Explanation: The access rights block (ARB) for the current process is too large to package and send to another subsystem.

User Action: Have your system manager use the Authorize Utility to remove unnecessary rights identifiers from the process and reenter the command.

ASUSPECT, customer defined text

Facility: LAVC, Local Area VAXcluster

Explanation: The local area VAXcluster network failure analysis has located a network problem and reported this network component as suspect.

User Action: Several PEDRIVER channels that were using this network component have failed. The analysis indicates that this component or something connected to it is likely to have caused the network problems. Have the system manager or network manager look into the network problem.

AUTONOTSTART, queue is autostart active, but not started

Facility: JBC, Job Controller

Explanation: You tried to start an autostart queue when none of its available nodes has autostart enabled.

User Action: Enter an ENABLE AUTOSTART[/QUEUES] command on the nodes in the queue's autostart node list.

BADCHECKSUM, message checksum failure

Facility: SYSTEM, VMS System Services

Explanation: A VAXcluster node has received a service request that contains user or object security profile information with questionable data integrity. The request cannot be serviced without potentially compromising system security.

User Action: Contact Digital Services or file a Software Performance Report (SPR).

BADFORMAT, format version mismatch in file 'filespec'

Facility: Shared by several facilities

Explanation: The specified file is not formatted as expected.

User Action: Verify that you specified the correct file.

BADGETJPI, unable to read process information

Facility: LAT, LAT Facility

Explanation: LATCP was unable to read process information before doing a SPAWN command.

User Action: Take appropriate action based on information in the accompanying message.

BADITMPROC, failed to process item code 'hex-number' correctly

Facility: QMAN, Queue Manager

Explanation: The queue manager encountered an internal error. When this error message occurs, a user request terminates with an INVITMCO error.

User Action: Submit a Software Performance Report (SPR) and include the item code number cited in the message.

BADLOGVER, transaction log file format version is unsupported

Facility: SYSTEM, VMS System Services

Explanation: The format of the transaction log file is not supported.

User Action: Use the LMCP facility to create a new transaction log file.

BAD_NAME, unable to repair log with invalid name format

Facility: LMCP, Log Manager Control Program

Explanation: A log repair could not be performed on the specified log file either because the file does not exist in the SYS\$JOURNAL directory or because the file is not named using the convention SYSTEM\$node-name.

User Action: Refer to Section B.11.1.1 and Section B.11.1.2 for information about naming and placing log files.

BADPARAM, parameter to DECthreads operation is invalid

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A parameter passed to a DECthreads routine is improper; for example, the value is of the wrong type or is out of range.

User Action: Determine which routine raised the exception. Then consult the documentation to determine the correct parameters and value ranges. Update your code accordingly and retry the operation.

BAD_SIZE, log file size invalid - permitted minimum is 100 blocks

Facility: LMCP, Log Manager Control Program

Explanation: You attempted to create a log file that is too small to use.

User Action: Recreate a log file specifying a file size of 100 blocks or more.

VMS System Messages

BUFTOOSMALL, request could not be completely satisfied due to limited buffer size

Facility: JBC, Job Controller

Explanation: Your \$GETQUI or \$SNDJBC request was not fully satisfied because the amount of information retrieved in response to the query exceeds the amount of data the queue manager can return in response to a single request.

User Action: Replace your large request with several smaller requests.

BUGCHECK, internal error detected in DECthreads

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: The DECthreads run-time program has discovered an internal inconsistency.

User Action: Run the program with the debugger. Use the SET OUTPUT LOG command so that the debugger outputs the results to a file. Type GO to run the program. When the bugcheck occurs, type SHOW CALLS. Submit a Software Performance Report (SPR) with the file produced by the debugger.

CANTATTACH, unable to ATTACH to target process

Facility: LAT, LAT Facility

Explanation: LATCP was unable to attach to the process specified by the ATTACH command for the reason described in an accompanying message.

User Action: Correct the situation based on the information in the accompanying message.

CANTCOPYSTR, unable to copy character string

Facility: LAT, LAT Facility

Explanation: An internal LATCP error occurred.

User Action: Take appropriate action based on information in the accompanying message.

CANTSPAWN, unable to SPAWN due to captive account

Facility: LAT, LAT Facility

Explanation: You cannot spawn out of LATCP using the SPAWN command when LATCP is being run from a captive account.

User Action: None.

CHANINTLK, channel usage interlocked

Facility: SYSTEM, VMS System Services

Explanation: An application attempted to perform a terminal I/O request to a pseudoterminal that has a virtual terminal linked to it.

User Action: Do not use the channel for terminal I/O until the virtual terminal is no longer linked.

CLOSEERR, error closing 'filespec'

Facility: Shared by several facilities

Explanation: The specified file cannot be closed. Usually an accompanying RMS message indicates why the file cannot be closed.

User Action: Take corrective action based on the accompanying message.

CMDERROR, error reported by command executor

Facility: LAT, LAT Facility

Explanation: The command executor module for LATCP cannot execute the command for the reason given in the accompanying message.

User Action: Correct the situation based on the information in the accompanying message.

CMDOBS, command obsolete - ignored

Facility: LAT, LAT Facility

Explanation: The specified LATCP command is obsolete; the command is ignored.

User Action: Discontinue use of this command.

CNTRSOBS, counters command qualifier obsolete - command ignored

Facility: LAT, LAT Facility

Explanation: The LATCP command SHOW COUNTERS is obsolete; the command is ignored.

User Action: Discontinue use of the SHOW COUNTERS command.

COMMERROR, unexpected error # 'number' in communicating with node CSID 'number'

Facility: QMAN, Queue Manager

Explanation: The queue manager encountered an internal error. The accompanying message provides more information about the cause of the error.

User Action: Submit a Software Performance Report (SPR) and include the message text. Create a copy of all *.QMAN\$JOURNAL* files using the BACKUP/IGNORE=INTERLOCK command as soon as possible and include the copy with the SPR, along with any SYS\$SYSTEM:QMAN\$QUEUE_MANAGER.DMP files. Also provide a copy of any messages written to the console or operator log file with the QUEUE_MANAGE or JOB_CONTROL username.

COMM_FAIL, DECdtm transaction manager communications failure prior to voting

Facility: DDTM, DECdtm Services

Explanation: The transaction aborted because of a communications failure between two DECdtm transaction managers involved in the transaction.

User Action: Check the network links between the nodes involved in the transaction.

CONATMPT, continuing attempts to connect to 'service-name'

Facility: LAT, LAT Facility

Explanation: A SET HOST/LAT connection failed and is being retried. This informational message is seen only if the /AUTOCONNECT qualifier is specified on SET HOST/LAT.

User Action: You can enter Ctrl/Y to cancel the retry.

VMS System Messages

CONFAIL, connection to 'service-name' not established

Facility: LAT, LAT Facility

Explanation: A SET HOST/LAT connection attempt was not successful.

User Action: Take appropriate action based on information in the accompanying message.

CONFQUAVAL, values for /DISCONNECT and /BREAK must be different

Facility: LAT, LAT Facility

Explanation: The SET HOST/LAT command specified the same character for both the /DISCONNECT and /BREAK qualifiers.

User Action: Specify different characters for the /DISCONNECT and /BREAK qualifiers.

CONLOST, connection to 'service-name' terminated

Facility: LAT, LAT Facility

Explanation: After a SET HOST/LAT connection was established, an error condition occurred that caused the connection to be abnormally terminated.

User Action: Determine the availability of the node that had the connection broken.

CONNECTED, session to 'service-name' on node 'node-name' established

Facility: LAT, LAT Facility

Explanation: The SET HOST/LAT session has been established to the specified service and node. No node name is specified in the message when the node is the same as the service.

User Action: None.

CONTIMEOUT, connection timed out, server not available, or incorrect server name

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt on a LAT device (LTAxxx:) failed when the connection request timed out. Either the remote node was not available or the LTA device describing the destination to receive the connection was set up incorrectly.

User Action: Check the mapping for the LTA device from which the connection was attempted or check to see whether the remote node described by the LTA device is available on the network.

CONTROLCL, operation completed under CTRL/C

Facility: LAT, LAT Facility

Explanation: The LATCP command completed after you entered Ctrl/C.

User Action: None.

CREPRCSTOP, failed to create a batch process; queue 'queue-name' will be stopped

Facility: QMAN, Queue Manager

Explanation: The queue manager could not create a process to execute a batch job. To avoid errors when trying to execute other batch jobs in the queue, the specified queue will be stopped upon completion of any jobs

currently executing in that queue. An accompanying message provides additional information.

User Action: Take action based on the accompanying message. Possibly there are too few process slots on the node. Correct the problem and try to restart the queue.

DATALOST, data lost

Facility: SYSTEM, VMS System Services

Explanation: Data was lost on a PTD\$WRITE operation because the terminal driver's type-ahead buffer is full.

User Action: Wait until the driver is ready for input and try entering the data again. For information on how to program the pseudoterminal, refer to the *VMS I/O User's Reference Manual: Part I*.

DATAOVERUN, data overrun

Facility: SYSTEM, VMS System Services

Explanation: This message can occur under the following conditions:

- More data has been read into the user buffer than the user buffer can hold.
- More data has been written into the user buffer than the user buffer can hold.
- Card reader data has been written into the controller data buffer before the driver has been able to receive previously sent data.
- Continued attempts to use PTD\$WRITE have resulted in data loss because the terminal driver's type-ahead buffer is full.

If this message is associated with a status code returned by a request to a magnetic tape driver, the data block read is longer than the assigned buffer. On a read reverse operation, the first data read and stored in the buffer is the data that was nearest the end-of-tape marker when the operation began.

User Action: There are several possible user actions:

- If there is too much data for the existing buffer, specify a larger buffer.
- If the problem occurred on a card reader operation, resubmit the cards to the reader.
- Turn on the alternate type-ahead buffer by using the DCL command SET TERMINAL/ALTYPEAHD. Then increase the type-ahead buffer size or the alternate type-ahead buffer size, or both, by modifying the TTY SYSGEN parameters.
- Wait until the driver is ready to receive input before you resume writing data. For information on how to handle flow control and the pseudoterminal, refer to the *VMS I/O User's Reference Manual: Part I*.

VMS System Messages

DEFER_Q_FULL, no space is currently available to process an AST request

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A call to a DECthreads service from an AST routine cannot be served immediately because there are too many outstanding requests.

User Action: AST routines using DECthreads are occurring too quickly. Reduce the number of requests or slow the rate of their arrival.

If you continue to have problems, submit a Software Performance Report (SPR) including a small test program that reproduces the problem.

DEFINEKEY, defined key 'key-name'

Facility: LAT, LAT Facility

Explanation: The specified key has been defined by LATCP.

User Action: None.

DELLINK, deleted link 'link-name'

Facility: LAT, LAT Facility

Explanation: The specified link has been deleted by LATCP.

User Action: None.

DELPOR, deleted port 'port-name'

Facility: LAT, LAT Facility

Explanation: The specified port has been deleted by LATCP.

User Action: None.

DELSERVICE, deleted service 'service-name'

Facility: LAT, LAT Facility

Explanation: The specified service has been deleted by LATCP.

User Action: None.

DISCONNECTED, session disconnected from 'service-name'

Facility: LAT, LAT Facility

Explanation: The SET HOST/LAT session has been disconnected.

User Action: None.

'virtual-unit:' does not contain the member named to VMB. System may not reboot.

Facility: OPCOM, Operator Communication Process

Explanation: Either of the following conditions can cause this message:

- The boot device is dismounted or failed out of the system disk shadow set.
- Shadowing finds the boot device missing from the system disk shadow set membership during any dismount operations on the system disk.

User Action: Mount the boot device back into the shadow set as soon as possible. If you cannot mount the boot device back into the shadow set, change the device name in VMB so the system can reboot.

DRIVERNOTSHUT, LATACP not initialized because driver not completely shut down

Facility: LAT, LAT Facility

Explanation: An attempt was made to run the LATACP process before the previous driver shutdown request had completed.

User Action: Wait until driver shutdown tasks have completed before attempting to start LATACP or determine whether some condition is preventing LAT driver shutdown from completing.

DUPCHARNAME, duplicate characteristic name

Facility: JBC, Job Controller

Explanation: A DEFINE/CHARACTERISTIC command specified a characteristic name that is already defined. Each characteristic must have a unique name.

User Action: Choose a name that is not yet defined or delete the old definition and redefine it.

DUPCHARNUM, duplicate characteristic number

Facility: JBC, Job Controller

Explanation: A DEFINE/CHARACTERISTIC command specified a characteristic number that is already defined. Each characteristic must have a unique number.

User Action: Choose a number that is not yet defined or delete the old definition and redefine it.

DUPFORMNAME, duplicate form name

Facility: JBC, Job Controller

Explanation: A DEFINE/FORM command specified a form name that is already defined. Each form must have a unique name.

User Action: Choose a name that is not yet defined or delete the old definition and redefine it.

END, control returned to node 'node-name'

Facility: LAT, LAT Facility

Explanation: The SET HOST/LAT session has ended.

User Action: None.

ENTNOTFOU, node/service entity not found

Facility: LAT, LAT Facility

Explanation: An attempt to locate information about a specified node or service ended with the local node finding no such information.

User Action: Check the network to ascertain that the specified node or nodes offering the specified service are available. Also, check that the group codes offered by the specified node or service coincide with the local node's user groups.

ERRCREKBD, unable to create virtual input device

Facility: LAT, LAT Facility

Explanation: LATCP cannot create a virtual input device for keyboard commands. An accompanying message explains why the virtual input device cannot be created.

User Action: Correct the situation based on the information in the accompanying message.

ERRVIRDPY, unable to create virtual output device

Facility: LAT, LAT Facility

Explanation: LATCP cannot create a virtual output device to display data. An accompanying message explains why the virtual output device cannot be created.

User Action: Correct the situation based on the information in the accompanying message.

EXCCOP, exception raised; VMS condition code follows

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: An exception has been raised by the EXC_HANDLING.H package, which provides portable exceptions for the C language. The accompanying VMS condition code identifies the error.

User Action: See the documentation for the software that your program is calling to determine the reason for this exception. Correct the problem or use the EXC_HANDLING.H package to provide an exception handler.

EXCEPTION, exception raised; address of exception object: 'object-address'

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: An exception has been raised by the EXC_HANDLING.H package, which provides portable exceptions for the C language.

User Action: See the documentation for the software that your program is calling to determine the reason for this exception. Correct the problem or use the EXC_HANDLING.H package to provide an exception handler.

EXISTENCE, object referenced does not currently exist

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A DECthreads routine has been requested to operate on an object that does not exist.

User Action: Consult the documentation for the DECthreads routine that issued this message to determine the conditions that caused it. Also check the program where the call is issued to determine which object or objects being passed as parameters do not currently exist.

EXIT_THREAD, current thread has been requested to exit

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: The `cma_exit_thread` routine has been called to force the thread to shut down in an orderly fashion. This message notifies all active exception handlers to perform any necessary cleanup activities.

User Action: None.

FAILOVER, attempting failover

Facility: LAT, LAT Facility

Explanation: A SET HOST/LAT connection to a service has been abnormally lost. SET/HOST LAT is attempting to connect to another node offering the same service.

User Action: None.

FATALERR, fatal error reading startup database

Facility: STDRV, System Startup Driver

Explanation: The startup driver encountered a fatal error while trying to read the startup database files. The startup operation is aborted. If this message occurs during a system reboot, VMS may not have been properly started.

User Action: Verify that the startup databases, defined by the logical names STARTUP\$STARTUP_VMS, STARTUP\$STARTUP_LAYERED, and STARTUP\$PHASES, are all available and can be accessed.

FREEDISK, free up 'number' blocks on disk 'disk-name'

Facility: QMAN, Queue Manager

Explanation: The specified amount of disk space is needed on the named disk.

User Action: Purge and delete files to make more space on the disk.

ILLPERNAM, 'string' is an illegal personal name

Facility: MAIL, Mail Utility

Explanation: You specified a string containing an illegal combination of characters, such as, for example, multiple consecutive spaces, special characters that MAIL cannot process, or unbalanced quotation marks.

User Action: Specify a personal name that avoids the problem.

INCDISABLED, incoming connections are currently disabled

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt failed because the driver is not accepting incoming LAT connections.

User Action: Enable incoming LAT connections (refer to the *VMS LAT Control Program (LATCP) Manual* for instructions) and retry the connection.

INCSHAMEM, system disk membership inconsistency

Facility: INIT, INITIALIZE Command

Explanation: The boot device is not currently a source member of the shadow set. One or more of the shadow set members named in the storage control block (SCB) of the boot device is inaccessible.

User Action: None.

INIALRPRO, DECthreads initialization is already in progress

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A call was made to the DECthreads initialization routine **cma_init** while DECthreads was still trying to initialize itself on a prior call. DECthreads initialization must complete before any DECthreads routines are used. Once DECthreads is fully initialized, all calls to **cma_init** complete successfully.

User Action: Remove the offending concurrent call to the **cma_init** routine or delay it until the first call to **cma_init** has completed.

INITFAIL, job controller initialization failure

Facility: JBC, Job Controller

Explanation: This message typically indicates that the system is improperly configured. The accompanying message provides more information.

User Action: Use AUTOGEN to reconfigure the system. If the problem does not seem to be associated with the system's configuration, submit a Software Performance Report (SPR) and include SYS\$SYSTEM:JBC\$JOB_CONTROL.DMP plus any messages written to the console or operator log file with the QUEUE_MANAGE or JOB_CONTROL username.

INSRES, insufficient resources to complete operation

Facility: LAT, LAT Facility

Explanation: The system does not have enough resources to service the user request.

User Action: Contact the system manager to determine which system resource is inadequate.

INTERNALERROR, internal error caused loss of process status

Facility: JBC, Job Controller

Explanation: A system error prevented the queue manager from obtaining the completion status of a process.

User Action: Ask your system manager to consult the operator log for messages associated with the process.

IN_USE, object referenced is already in use

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: The DECthreads operation cannot be performed on the specified object because it is already in use; for example, the routine is attempting to delete a mutex that is locked.

User Action: Determine which routine caused the error and make sure the object is in an appropriate state before attempting the operation.

INVBUSNAM, invalid Local Area VAXcluster BUS name

Facility: SYSTEM, VMS System Services

Explanation: An invalid BUS name was specified when calling the SYS\$LAVC_START_BUS or SYS\$LAVC_STOP_BUS routine.

User Action: Check the BUS name to make sure it contains at least three ASCII characters to specify the LAN device to be used. For example, XQA is a valid BUS name for the device name _XQA0:. The full device name _XQA0: is also a valid BUS name.

INVCMD, invalid command

Facility: LAT, LAT Facility

Explanation: The specified LATCP command was invalid for the reason given in an accompanying message.

User Action: Correct the situation based on the information in the accompanying message.

INVCOMPID, invalid component ID

Facility: SYSTEM, VMS System Services

Explanation: An invalid component identification (ID) value was specified in the network path description. SYS\$LAVC_DEFINE_NET_PATH returns the invalid component ID value in the buffer provided for BAD_COMPONENT_ID.

User Action: A coding error occurred in the user program that passed the component ID value. Review how the component ID value was obtained and why it was placed into the network component list. Component IDs are valid only on the local system and are not valid across system boots or after calling SYS\$LAVC_DISABLE_ANALYSIS.

INVCOMPLIST, invalid component list

Facility: SYSTEM, VMS System Services

Explanation: The network component list used when calling SYS\$LAVC_DEFINE_NET_PATH was built incorrectly.

User Action: Check the network component list to make sure it contains the component identification (ID) values for two NODE components and two ADAPTER components. The first and last component ID values should correspond to NODE components.

INVCOMPTYPE, invalid component type

Facility: SYSTEM, VMS System Services

Explanation: An invalid component type value was passed to SYS\$LAVC_DEFINE_NET_COMPONENT.

User Action: Use one of the component type values defined by the macro \$PEMCOMPDEF: COMP\$C_NODE, COMP\$C_ADAPTER, COMP\$C_COMPONENT, or COMP\$C_CLOUD.

INVJOURNAL, invalid data found in job journal file

Facility: QMAN, Queue Manager

Explanation: The queue manager could not identify a piece of data in the job journal file.

User Action: Submit a Software Performance Report (SPR). Create a copy of all *.QMAN\$JOURNAL* files using the BACKUP/IGNORE=INTERLOCK command as soon as possible and include the copy with the SPR. Also provide a copy of any messages written to the console or operator log file with the QUEUE_MANAGE or JOB_CONTROL username.

INVPREFIX, invalid prefix format string - check FAO directives

Facility: CLI, Command Language Interpreter (DCL)

Explanation: The prefix format string specified with SET PREFIX is invalid for one of the following reasons: it is too long; it has invalid FAO directives; or it has FAO directives that are not supported with this command.

User Action: Check the SET PREFIX description in Section 3.4.1 for restrictions on the format string contents and resulting length. Modify the string accordingly and retry the command.

INVQMANMSG, queue manager received an improper message

Facility: QMAN, Queue Manager

Explanation: The queue manager encountered an internal error.

User Action: Submit a Software Performance Report (SPR). Create a copy of all *.QMAN\$JOURNAL* files using the BACKUP/IGNORE=INTERLOCK command as soon as possible and include the copy with the SPR, along with any SYS\$SYSTEM:QMAN\$QUEUE_MANAGER.DMP files. Also provide a copy of any messages written to the console or operator log file with the QUEUE_MANAGE or JOB_CONTROL username.

INVREF, invalid object reference

Facility: QMAN, Queue Manager

Explanation: The queue manager encountered an internal error.

User Action: Submit a Software Performance Report (SPR). Create a copy of all *.QMAN\$JOURNAL* files using the BACKUP/IGNORE=INTERLOCK command as soon as possible and include the copy with the SPR, along with any SYS\$SYSTEM:QMAN\$QUEUE_MANAGER.DMP files. Also provide a copy of any messages written to the console or operator log file with the QUEUE_MANAGE or JOB_CONTROL username.

INVSECDOMAIN, request originated outside of local security domain

Facility: SYSTEM, VMS System Services

Explanation: A VAXcluster node has received a service request containing user or object security profile information that originates outside the security domain of the receiving node. The request cannot be serviced without potentially compromising system security.

User Action: Make sure that all VAXcluster nodes refer to the same rights database file (SYS\$SYSTEM:RIGHTSLIST.DAT).

ITMREMOVED, meaningless items were removed from request

Facility: JBC, Job Controller

Explanation: You specified one or more item codes or qualifiers that are not meaningful in this command. The command is processed and the meaningless items are ignored.

User Action: Determine which item codes and qualifiers are meaningless in this command and discontinue using them in this context.

JOBDELFO, job 'job-name' (entry 'number' for user 'username') deleted during queue manager failover

Facility: QMAN, Queue Manager

Explanation: The queue manager detected corruption in the specified job and deleted the job.

User Action: Notify the user that the specified job was deleted. The user might want to resubmit the job.

Submit a Software Performance Report (SPR). Create a copy of all *.QMAN\$JOURNAL* files using the BACKUP/IGNORE=INTERLOCK command as soon as possible and include the copy with the SPR. Also provide a copy of any messages written to the console or operator log file with the QUEUE_MANAGE or JOB_CONTROL username.

JOBNOTEXEC, specified job is not executing

Facility: JBC, Job Controller

Explanation: You used STOP/ENTRY or STOP/ABORT to try to abort a job that was not being processed at the time.

User Action: Use DELETE/ENTRY to eliminate the job.

LATSTOPPING, LAT shutdown in progress

Facility: LAT, LAT Facility

Explanation: The LAT software has been stopped on the local node.

User Action: None.

LISTTOOSHORT, itemlist too short

Facility: LAT, LAT Facility

Explanation: A LAT SETMODE \$QIO request cannot be fulfilled because the specified item list is not large enough.

User Action: Increase the size of the item list specified in the \$QIO request.

LOG_IN_USE, unable to locate current end of file - dump aborted

Facility: LMCP, Log Manager Control Program

Explanation: The attempted log file dump aborted when LMCP was unable to locate the end of the log file because the system was too busy (transaction start rate was too high).

User Action: Try the dump again when the system is less active.

LOSTINFO, information for 'name' was lost due to database corruption

Facility: QMAN, Queue Manager

Explanation: The queue manager detected corruption in the definition of the specified queue, form, or characteristic. The corrupt information was deleted.

User Action: Review the full definition of the affected object and correct the definition to include the deleted information.

Submit a Software Performance Report (SPR). Create a copy of all *.QMAN\$JOURNAL* files using the BACKUP/IGNORE=INTERLOCK command as soon as possible and include the copy with the SPR. Also provide a copy of any messages written to the console or operator log file with the QUEUE_MANAGE or JOB_CONTROL username.

LOSTMSG, broadcast message was lost

Facility: LAT, LAT Facility

Explanation: LATCP encountered an error while trying to display a broadcast message.

User Action: None.

LOWDISKSPACE, disk space is low on 'disk-name'

Facility: QMAN, Queue Manager

Explanation: The queue manager is still progressing through its work, but a lack of disk space has been detected. This message indicates a potential problem if preventive action is not taken. The accompanying FREEDISK message provides details.

User Action: Purge and delete files to make more space on the disk.

LOWMEMORY, the queue manager process may require more virtual memory than is currently available

Facility: QMAN, Queue Manager

Explanation: The queue manager is still progressing through its work, but a lack of virtual memory has been detected. This message indicates a potential problem if preventive action is not taken.

User Action: You might need to run AUTOGEN to increase the SYSGEN parameter VIRTUALPAGECNT.

LRJACCESSDENIED, access denied

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because access was denied.

User Action: Check group codes of the local node against group codes of the object node.

LRJACCESSREJECT, immediate access is rejected

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because immediate access was rejected.

User Action: Retry the connection later.

LRJCORRUPT, corrupted request

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because of a corrupted network message.

User Action: Retry the connection. Contact the network manager if problems persist.

LRJDELETED, queue entry deleted by server

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the connection request was deleted from the local queue at the object node.

User Action: Retry the connection later.

LRJDISABLE, service is disabled

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the object service is disabled.

User Action: Enable the object service and retry the connection.

LRJILLEGAL, illegal request parameters

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the object node detected illegal request parameters or an inconsistency in a LAT command message received from the local node.

User Action: Submit a Software Performance Report (SPR).

LRJINUSE, port or service in use

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the object port or object service is in use.

User Action: Retry the connection when the object service or object port becomes available.

LRJIVPASSWORD, invalid service password

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the object service password is invalid.

User Action: Retry the connection using the correct service password.

LRJNAMEUNKNOWN, port name is unknown

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the name of the object port you are trying to connect to is unknown.

User Action: Verify the object port name and retry the connection.

LRJNOSERVICE, no such service

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the specified object service does not exist.

User Action: Verify that the object service name is correct and retry the connection.

LRJNOSTART, session cannot be started

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the session cannot be started.

User Action: Try the connection again later.

LRJNOTOFFERED, service is not offered on the requested port

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the object service is not offered on the requested port.

User Action: Specify a port that offers the desired service.

LRJNOTSUPPORT, requested function is not supported

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because the object node has detected an unsupported command operation message code in a LAT command message received from the local node.

User Action: Submit a Software Performance Report (SPR).

LRJRESOURCE, insufficient resources at server

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because of insufficient resources on the object node.

User Action: Retry the connection later.

LRJSHUTDOWN, system shutdown in progress

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established because a system shutdown of the object node is in progress.

User Action: None.

LRJUNKNOWN, unknown

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the connection cannot be established for an unknown reason.

User Action: None.

LRJUSERDIS, user requested disconnect

Facility: LAT, LAT Facility

Explanation: This LAT rejection message indicates that the session was normally disconnected from the object node.

User Action: None.

MAXLINKS, maximum links already defined

Facility: LAT, LAT Facility

Explanation: The link cannot be created because the maximum number of allowable links is already defined.

User Action: None.

MAXSERV, maximum number of services exceeded

Facility: LAT, LAT Facility

Explanation: You attempted to create more than 255 services on the local node.

User Action: You must delete a service in order to add one.

MODLINK, modified link 'link-name'

Facility: LAT, LAT Facility

Explanation: The specified link has been modified by LATCP.

User Action: None.

MODNODE, modified characteristic(s) of local node

Facility: LAT, LAT Facility

Explanation: Characteristics of the local node have been modified by LATCP.

User Action: None.

MODPORT, modified port 'port-name'

Facility: LAT, LAT Facility

Explanation: The specified port has been modified by LATCP.

User Action: None.

MODSERVICE, modified service 'service-name'

Facility: LAT, LAT Facility

Explanation: The specified service has been modified by LATCP.

User Action: None.

MSGNGENDS, missing or misspelled ENDSUBROUTINE statement detected while scanning for label

Facility: CLI, Command Language Interpreter (DCL)

Explanation: A SUBROUTINE command with no ending ENDSUBROUTINE command or with a misspelled ENDSUBROUTINE command was detected while executing a CALL command. This condition can prevent the CALL command from locating an existing destination label.

User Action: Check the command procedure for one or more missing or misspelled ENDSUBROUTINE commands; correct as necessary.

NAMETOOLONG, link name is too long

Facility: LAT, LAT Facility

Explanation: You attempted to create or set a link with a name longer than 16 characters.

User Action: Use link names of 16 or fewer characters.

NEWLINK, created link 'link-name'

Facility: LAT, LAT Facility

Explanation: The specified link has been created by LATCP.

User Action: None.

NEWLOGNAME, created logical name 'logical-name' in table 'table-name'

Facility: LAT, LAT Facility

Explanation: The specified logical name has been created by LATCP in the specified table.

User Action: None.

NEWPORT, created port 'port-name'

Facility: LAT, LAT Facility

Explanation: The specified port has been created by LATCP.

User Action: None.

NEWSERVICE, created service 'service-name'

Facility: LAT, LAT Facility

Explanation: The specified service has been created by LATCP.

User Action: None.

NOACP, no LATACP to process request

Facility: LAT, LAT Facility

Explanation: You requested information from the local LAT software but LATACP is not currently running on the local node.

User Action: Check to see whether LAT startup has executed correctly or if LAT shutdown has been performed on the local node.

NOALOCCLASS, allocation class not allowed with shadowing phase II virtual unit name

Facility: MOUNT, Mount Utility

Explanation: An allocation class was specified in the name of the virtual unit. Allocation classes are not allowed in virtual unit names with volume shadowing phase II (VMS Volume Shadowing).

User Action: Reenter the command without specifying an allocation class on the virtual unit. The virtual unit must be specified in the form DSA or DSA $nnnn$, where $nnnn$ represents a unique number from 0 to 9999.

NOAUTOSTART, node does not have the autostart feature enabled

Facility: JBC, Job Controller

Explanation: You entered a DISABLE AUTOSTART [/QUEUES] command for a node on which the autostart feature is not currently enabled.

User Action: None.

NOCOMPLSTS, no component lists are defined

Facility: SYSTEM, VMS System Services

Explanation: No component lists were defined using calls to SYS\$LAVC_DEFINE_NET_PATH before calling SYS\$LAVC_ENABLE_ANALYSIS. As a result, no data is available to perform the local area VAXcluster network failure analysis.

User Action: Perform the following steps to properly define the network description before calling SYS\$LAVC_ENABLE_ANALYSIS:

1. Call SYS\$LAVC_DEFINE_NET_COMPONENT for each network component.

2. Describe each network path used by building a list from the component identifications (IDs) returned by SYS\$LAVC_DEFINE_NET_COMPONENT.
3. Call SYS\$LAVC_DEFINE_NET_PATH to define the network component lists.
4. After all the network paths are defined, call SYS\$LAVC_ENABLE_ANALYSIS to enable the local area VAXcluster network failure analysis.

NOCURTID, no process default transaction currently defined

Facility: SYSTEM, VMS System Services

Explanation: The user program attempted to terminate a default transaction when none was defined.

User Action: Correct the program so that it specifies a transaction identifier (TID).

NODECNTRSONLY, only counter information is available for this node

Facility: LAT, LAT Facility

Explanation: The specified node offers no services known to the local node. However, there is a connection from the node and counter information is maintained.

User Action: None.

NODESHUT, node shutdown in progress

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt on an application port or dedicated port was rejected because the node state is Shut.

User Action: Wait until the node is in the On state to make new connections. Refer to the *VMS LAT Control Program (LATCP) Manual* for a full description of the node states.

NODEVINFO, unable to retrieve device information on 'disk-name'

Facility: QMAN, Queue Manager

Explanation: The queue manager received a bad return value from a call to the \$GETDVI system service. The accompanying message provides information about why the operation failed.

User Action: Take corrective action based on the accompanying message.

NODISKSPACE, disk space not available for queue manager to continue

Facility: QMAN, Queue Manager

Explanation: The queue manager cannot process any queuing requests because of a lack of disk space. The accompanying FREEDISK message provides details.

User Action: Purge and delete files to make more space on the disk.

NODUNAV, node 'node-name' not currently reachable

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt to a specified service and node failed because the specified node is not currently reachable.

User Action: Retry the connection or determine why the remote node is not currently reachable.

VMS System Messages

NOIDBAVAIL, unable to allocate an IDB

Facility: LAT, LAT Facility

Explanation: LATCP cannot allocate enough virtual memory for an internal data structure needed to execute a command. An accompanying message explains why the virtual memory cannot be allocated.

User Action: Correct the situation based on information in the accompanying message.

NOINFO, no information in database

Facility: NCP, Network Control Program

Explanation: An NCP command (usually a SET command) was executed when there was no data to act upon in the database. This error commonly occurs during system startup when a SET KNOWN *component* ALL command is executed and there is no data to be copied into the volatile database from the permanent database. If the error occurs during system operation, a command has attempted to manipulate data that does not exist; for example, a command specifies a nonexistent component.

User Action: Ignore this error if it occurs during system startup. If you receive this error during system operation, reissue the command specifying an existing system component.

NOITMLST, unable to allocate virtual memory for command itemlist

Facility: LAT, LAT Facility

Explanation: LATCP cannot allocate enough virtual memory for a LAT item list needed to execute a command. An accompanying message explains why the virtual memory cannot be allocated.

User Action: Correct the situation based on information in the accompanying message.

NOMEANING, qualifiers 'qualifier-names' are no longer meaningful for the 'command-name' command

Facility: Shared by several facilities

Explanation: The command contains one or more DCL qualifiers that have been phased out in a new release of VMS.

User Action: Check release notes or new documentation for updated information about the specified command.

NOMORENODS, no more nodes in database

Facility: LAT, LAT Facility

Explanation: This informational message is returned when a wildcard search of nodes in the database reaches the last node.

User Action: None.

NOMORESVCS, no more services in database

Facility: LAT, LAT Facility

Explanation: This informational message is returned when a wildcard search of services in the database reaches the last service.

User Action: None.

NONODE, node name has not been initialized

Facility: LAT, LAT Facility

Explanation: The local LAT node has not been initialized.

User Action: Use the LATCP command SET NODE to initialize the local LAT node.

NOPROCTPS, no transaction structures for this process

Facility: SDA, System Dump Analyzer

Explanation: The selected process is not a participant in any active transactions.

User Action: None.

NOREADER, no read channel is assigned to the device

Facility: SYSTEM, VMS System Services

Explanation: This message can be returned under either of the following conditions:

- A sensemode readercheck \$QIO request or a write readercheck \$QIO request was issued to a mailbox that has no reader assigned to it.
- A write readercheck \$QIO request was issued to a mailbox when no read channels were assigned to the mailbox.

User Action: The mailbox driver allows channels to be assigned to the mailbox as read-only, write-only, or read/write (the default). Applications using read-only or write-only channels should anticipate this error and count on it for synchronization. If necessary, recode your application to expect this error or consider using the older mailbox driver features; that is, use read/write channels and do not use readercheck on a write request. Refer to the *VMS I/O User's Reference Manual: Part I* for more information about the mailbox driver.

NOREMBROAD, no VAXcluster terminals were notified because OPCOM is not available

Facility: OPCOM, Operator Communication Process

Explanation: A REPLY command attempting to send a message to terminals on other nodes within a VAXcluster has failed because OPCOM is not available to satisfy the request. The message is sent only to terminals on the local node.

User Action: Restart OPCOM with the following command:

```
$ @SYSS$SYSTEM:STARTUP OPCOM
```

NOREMWAIT, /WAIT requested, therefore no VAXcluster terminals notified

Facility: OPCOM, Operator Communication Process

Explanation: A REPLY command attempted to send a message to terminals on other nodes within a VAXcluster, but the /WAIT qualifier was specified, which requests that the message be sent synchronously.

User Action: If the message must be delivered to terminals on other VAXcluster nodes, reissue the command without the /WAIT qualifier.

NOSELF, connecting to the local node is not allowed

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt to a service offered by the local node was targeted to the local node.

User Action: Retry the connection and specify another node that offers the service.

NOSRVC, service 'service-name' not known

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt was issued for a specified service that is unknown to the local node.

User Action: Retry the connection until the service becomes known or see if there is something wrong with a node offering the specified service. Possibly the service group codes on the local node do not intersect with the group codes for the specified service.

NOSTACKMEM, no space is currently available to create a new stack

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A call to **cma_create_thread** or another DECthreads routine requires a new stack to be created, but there is insufficient space to create it.

User Action: Reduce the value of the stack size attribute so that it does not exceed the stack cluster size.

NOSUCHID, no such identifier

Facility: SYSTEM, VMS System Services or AUTHORIZE, Authorize Utility

Explanation: Either the translation failed or the rights database has no record of the identifier. You must add an identifier to the rights database before you can use the VMS Authorize Utility or you must add an identifier to one of the security system services to grant the identifier to or revoke it from a user. The message occurs if the identifier or the user to whom you are granting the identifier does not exist.

User Action: Check the spelling of the identifier. Use the AUTHORIZE command SHOW/IDENTIFIER to determine whether the identifiers exist. Add any missing identifier using the AUTHORIZE command ADD /IDENTIFIER.

NOSUCHNODE, node 'node-name' not known

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt to a specified service and node failed because the target node name is unknown.

User Action: Retry the connection until the specified node is known or see if a problem with the specified node is preventing its network visibility. Possibly the local node's service group codes do not intersect with the specified remote node's group codes.

NOSYSCLF, no common logging structures

Facility: SDA, System Dump Analyzer

Explanation: There are no transaction logs currently open on this node.

User Action: None.

NOSYSIPC, no IPC structures

Facility: SDA, System Dump Analyzer

Explanation: There is no IPC activity currently on this node.

User Action: None.

NOSYSTPS, no transaction structures

Facility: SDA, System Dump Analyzer

Explanation: There are no active transactions currently on this node.

User Action: None.

NOTALLREQUE, all jobs in source queue could not be requeued to target queue

Facility: JBC, Job Controller

Explanation: Some of the jobs specified in an ASSIGN/MERGE command were not suitable for execution on the specified target queue.

User Action: Enter a SET ENTRY/REQUEUE=*queue-name* command to requeue the jobs remaining in the source queue to a queue that has the necessary settings to execute those jobs.

NOTATERM, command device is not a terminal

Facility: LAT, LAT Facility

Explanation: You attempted to use SET HOST/LAT from a device that is not a terminal.

User Action: Use SET HOST/LAT from a terminal device only.

NOTCMSTACK, the current stack was not allocated by DECthreads

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: The program attempted to call a DECthreads routine while the thread stack pointer register held an address in a stack that was not allocated by DECthreads. Because DECthreads uses the value in the thread stack pointer register to determine which thread is currently running, all calls to DECthreads routines must be performed on a stack that was allocated by DECthreads.

User Action: Modify the program so that it does not switch stacks, or call DECthreads to create an additional stack and assign it to the thread.

NOTDISM, unable to dismount 'device-id'

Facility: BACKUP, Backup Utility

Explanation: The Backup Utility cannot dismount a tape drive specified by the command line qualifier /RELEASE_TAPE.

User Action: An accompanying message indicates the type of user action required, if any.

NOTEXIST, folder 'folder-name' does not exist

Facility: MAIL, Mail Utility

Explanation: The command cannot be executed because it specifies a folder that does not exist.

User Action: Use the MAIL command DIRECTORY/FOLDER to display a list of existing folders. Then retry the command using an existing folder name.

NOTLOADED, LAT terminal port driver (LTDRIVER) is not loaded

Facility: LAT, LAT Facility

Explanation: You attempted to execute LATACP when LTDRIVER was not loaded.

User Action: Check to see whether the LAT software is properly configured.

NOTMEANINGFUL, specified item code is no longer meaningful

Facility: JBC, Job Controller

Explanation: The specified item code once affected the results of the command, but it no longer does so.

User Action: Discontinue using this item code with this command.

NOTMODEM, VAX/VMS host system modem not wired correctly - contact your system manager

Facility: LOGIN, Login Processor

Explanation: The terminal line is set to /MODEM and TTDRIVER did not detect all the necessary modem signals within 30 seconds of a login attempt.

User Action: For information on how the TTDRIVER identifies a valid modem line, refer to the section on modem control of terminal drivers in the *VMS I/O User's Reference Manual: Part I*. Make sure that the following conditions are met:

- The modem cable connecting the modem provides the correct signal.
- The terminal port supports modem use.
- The modem provides the correct signals in the correct order.

NOTOFFERED, service not offered by requested node

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt to a specified service for a specified node failed because the node does not offer the selected service.

User Action: Retry the connection and specify a node that offers the desired service.

NOTPSHARE, shareable image for DECdtm Services SDA support unavailable

Facility: SDA, System Dump Analyzer

Explanation: The shareable image for DECdtm services is not installed on this node.

User Action: Ensure that shareable image SYS\$SHARE:SDATP\$SHARE.EXE is installed before executing any SDA commands.

NOTSUPPORTED, specified item code or function code is not supported

Facility: JBC, Job Controller

Explanation: You attempted to use a new feature on a node that has not been upgraded.

User Action: Upgrade the node before attempting to specify the new item code or function code.

NOTWITHCONN, parameter cannot be modified with connections active or pending

Facility: LAT, LAT Facility

Explanation: An attempt to modify a LAT parameter failed because that parameter cannot be modified while active connections exist or while a connection request is pending. For a list of parameters that cannot be changed with connections active or pending, refer to the *VMS LAT Control Program (LATCP) Manual*.

User Action: Wait until there are no outstanding connection requests before modifying the parameter.

NOWRITER, no write channel is assigned to the device

Facility: SYSTEM, VMS System Services

Explanation: This message can be returned under either of the following conditions:

- A sensemode writercheck \$QIO request or a read writercheck \$QIO request was issued to a mailbox that has no writer assigned to it.
- A read writercheck \$QIO request was issued to a mailbox when no write channels were assigned to the mailbox.

User Action: The mailbox driver allows channels to be assigned to the mailbox as read-only, write-only, or read/write (the default). Applications using read-only or write-only channels should anticipate this error and count on it for synchronization. If necessary, recode your application to expect this error or consider using the older mailbox driver features; that is, use read/write channels and do not use writercheck on a read request. Refer to the *VMS I/O User's Reference Manual: Part I* for more information about the mailbox driver.

OPENERR, error opening 'filespec'

Facility: Shared by several facilities

Explanation: The specified file cannot be opened. Usually an accompanying RMS message indicates why the file cannot be opened.

User Action: Take corrective action based on the accompanying message.

OPENFAIL, failure opening component file 'file-number', 'file-name'

Facility: STDRV, System Startup Driver

Explanation: The startup driver failed to open one of the system files that describes the tasks that need to be performed at startup time. The startup operation attempts to continue, but may not properly perform all startup tasks.

User Action: Make sure that the named file is available and can be read.

OPINPROG, previously requested operation is incomplete

Facility: SYSTEM, VMS System Services

Explanation: A request could not be completed because of outstanding requests on the service.

User Action: Submit a Software Performance Report (SPR) that describes the conditions leading to the error. Include a BACKUP save set containing the output of both the LMCP DUMP command and the DCL DUMP command for the log file.

ORBTTOOBIG, object rights block too big

Facility: SYSTEM, VMS System Services

Explanation: The object rights block (ORB) for the specified object is too large to package and send to another subsystem.

User Action: Have your system manager use the ACL editor to remove unnecessary access control lists (ACLs) from the object or reorganize the ACLs. See the *Guide to VMS System Security* for more information about ACLs.

OUTOFRANGE, value specified is not within the legal range for this qualifier

Facility: LAT, LAT Facility

Explanation: You specified a value that is out of range for a LATCP qualifier.

User Action: Specify a value within the legal range (see the *VMS LAT Control Program (LATCP) Manual*).

PRIOSMALL, scheduling priority has smaller value than requested

Facility: JBC, Job Controller

Explanation: A user without ALTPRI or OPER privilege specified a value for a job's priority that exceeded the queue's maximum priority for nonprivileged users. The job is entered in the queue, but its scheduling priority is lower than the value requested by the user.

User Action: Use SHOW ENTRY/FULL to see the priority assigned to the job. If you must specify a higher scheduling priority, acquire the necessary privileges and use the DCL command SET ENTRY/PRIORITY to modify the job's priority, or see your system manager.

PSUSPECT, customer defined text

Facility: LAVC, Local Area VAXcluster

Explanation: The local area VAXcluster network failure analysis has located a network problem and reported this network component as the primary suspect.

User Action: Several PEDRIVER channels that were using this network component have failed. The analysis indicates that this component or something connected to it is the most likely cause of the network problems. Have the system manager or network manager look into the network problem.

QMANCREPRC, queue manager process could not be created

Facility: JBC, Job Controller

Explanation: The job controller could not create a queue manager process. An accompanying message gives information about why the process could not be created. One possible cause is too few process slots on the node.

User Action: Correct the problem described in the accompanying message and try to restart the queue manager.

QMANDEL, unexpected queue manager process termination

Facility: JBC, Job Controller

Explanation: A queue manager process exited without being requested to do so. An accompanying message gives information about why the process terminated.

User Action: Take action based on the accompanying message.

QMANNOTSTARTED, queue manager could not be started

Facility: JBC, Job Controller

Explanation: A START/QUEUE/MANAGER request failed to complete successfully.

User Action: Check the console or operator log file for messages from the JOB_CONTROL or QUEUE_MANAGE username explaining why the queue manager could not be started. If you included the directory specification with the START/QUEUE/MANAGER command, verify that you specified the correct directory.

QUALOBS, qualifier obsolete - '/qualifier' ignored

Facility: LAT, LAT Facility

Explanation: The specified LATCP command qualifier is obsolete and has no effect. The rest of the command is executed.

User Action: Do not specify this qualifier in future commands.

QUEAUTOOFF, queue 'queue-name' is now autostart inactive

Facility: QMAN, Queue Manager

Explanation: The specified autostart queue has been stopped without a user request. The queue manager will not restart the queue until a user enters the START/QUEUE command for the queue.

User Action: An accompanying message explains why the queue stopped. Correct the problem and try to restart the queue.

QUEDISABLED, disabled queue cannot be modified, nor can jobs be submitted to it

Facility: JBC, Job Controller

Explanation: The queue manager disabled the queue upon detection of database corruption.

User Action: Ask the system manager to delete and recreate the queue to which your command was directed.

QUENOTMOD, modifications not made to running queue

Facility: JBC, Job Controller

Explanation: You tried to change a feature of the queue that can be changed only when the queue is in the stopped state.

User Action: Enter a STOP/QUEUE/RESET or STOP/QUEUE/NEXT command, then reenter your original command when the queue is in the stopped state.

QUENOTSTART, queue 'queue-name' could not be started on node 'node-name'

Facility: QMAN, Queue Manager

Explanation: An error occurred while trying to start the specified autostart queue on the specified node.

User Action: An accompanying message explains why the operation failed. Correct the problem and try to start the queue again.

QUOTAFNF, quota file not found on volume

Facility: MOUNT, Mount Utility

Explanation: The MOUNT command specified /QUOTA or /CACHE=QUOTA, but there is no quota file on the volume.

User Action: Create a quota file on the volume using the DISKQUOTA or the SYSMAN utility.

REFERENCED, existing references prevent deletion

Facility: JBC, Job Controller

Explanation: Existing references to the specified form, characteristic, or queue by other queues or jobs prevent the specified item from being deleted.

User Action: Use the SHOW QUEUE/FULL/ALL command to locate all such references. Remove the existing references and retry the delete operation.

REINITERR, error attempting reinitialization

Facility: LAT, LAT Facility

Explanation: LATCP cannot reinitialize in order to accept another command. An accompanying message explains why the program cannot reinitialize.

User Action: Correct the situation based on the information in the accompanying message.

RMALRDCL, resource manager name has already been declared

Facility: SYSTEM, VMS System Services

Explanation: This message indicates an error in the resource manager.

User Action: Submit a Software Performance Report (SPR) that describes the conditions leading to the error. Include a BACKUP save set containing the output for both the LMCP DUMP command and the DCL DUMP command for the log file.

RMTPATH, description of path between two remote nodes

Facility: SYSTEM, VMS System Services

Explanation: The described network path represents a network path between two remote nodes instead of a path used by the local node. This network path is not necessary for the local area VAXcluster network failure analysis performed by the local node.

User Action: Removing this network path definition will prevent this informational message from occurring. However, this action is optional.

SCRATCH_HEADER, scratch header used by XQP Movefile operation

Facility: ANALDISK, Analyze/Disk_Structure Utility

Explanation: The Analyze/Disk_Structure Utility found a scratch file header (a temporary file header used by Movefile). This condition can be reported while an ANALYZE/DISK_STRUCTURE operation is being performed.

During a Movefile operation, blocks can be temporarily allocated to more than one file header. In such cases, this message can be accompanied by one or more MULTALLOC messages. These messages cease when the scratch header is released.

User Action: If the message occurs while you are performing an ANALYZE /DISK_STRUCTURE/NOREPAIR operation on a disk that is in use, no action is required.

If the message occurs while you are analyzing a disk after a system crash, release any scratch file headers on the disk by performing an ANALYZE /DISK_STRUCTURE/REPAIR or SET VOLUME/REBUILD operation on the disk.

SEG_FAIL, process failed prior to voting

Facility: DDTM, DECdtm Services

Explanation: The transaction was aborted because a process or image within the transaction failed.

User Action: Retry the transaction after the problem with the process or image has been corrected.

SERUNAV, service 'service-name' not currently available

Facility: LAT, LAT Facility

Explanation: A LAT connection was attempted to a service that is known by the local node but that is not currently available.

User Action: Determine the availability problem with the remote node offering the specified service. Possibly the specified service has disabled connection requests.

SERVEXISTS, service name already exists

Facility: LAT, LAT Facility

Explanation: You attempted to create a service using the name of a service that already exists on the local node.

User Action: Create a service using a different name.

SESLIM, session limit reached

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt failed because the current LAT session limit has already been reached.

User Action: Use the LATCP command SET NODE/SESSION_LIMIT=OUTGOING to increase the session limit or wait for a session slot to become available. Refer to the *VMS LAT Control Program (LATCP) Manual* for more information.

SHADBOOTFAIL, shadowing failed to boot from system disk shadow set

Facility: BUGCHECK, System Bugcheck

Explanation: Any of the following conditions can cause this error:

- A failure to allocate memory.
- One or more critical devices is inaccessible.
- The boot device is the target of a full copy operation.
- The boot device is not a source member of the existing shadow set.

User Action: Try one or more of these user actions:

- If the boot device is the target of a full copy operation or is not a source member of the existing shadow set, change the device name in VMB to be a source member and reboot the node.
- If the boot device is a source member of the existing shadow set, check the booting device's connections to all other shadow set members.
- If all device and system connections are fine, check the SYSGEN parameter settings for inappropriate memory configurations.

SHADDETINCON, SHADOWING detects inconsistent state

Facility: BUGCHECK, System Bugcheck

Explanation: The volume shadowing software reached an unrecoverable or inconsistent situation because the software failed an internal inconsistency check.

User Action: Submit a Software Performance Report (SPR) that describes the conditions leading to the error. If the system is configured to produce a memory dump, include the dump file with the SPR.

SHASINGMBR, single member system shadow set formed

Facility: INIT, INITIALIZE Command

Explanation: The shadow set membership is changing to form a single-member shadow set consisting of only the boot device.

User Action: None.

SIGNAL_Q_FULL, unable to process condition variable signal

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A call to the `pthread_cond_signal_int_np` or `cma_cond_signal_int` routine cannot be performed because there are too many outstanding requests.

User Action: Calls to the `cma_cond_signal_interrupt` routine are occurring too frequently. Reduce the number of calls or slow the rate of their arrival.

SRCMEM, only source member of shadow set cannot be dismounted

Facility: DISMOUNT, DISMOUNT Command

Explanation: An attempt was made to dismount a shadow set member that was the only valid source member of the set.

User Action: If there is only one shadow set member, it cannot be dismounted. To dissolve the shadow set, dismount the virtual unit. If there is more than one member, remove a full member and wait for copy operations to complete before dismounting a member.

SRVCNODE, service 'service-name' not offered by node 'node-name'

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt to a specified service and node failed because the node does not offer the specified service.

User Action: Retry the connection request and specify a node that offers the desired service.

SRVDIS, outgoing connections are disabled

Facility: LAT, LAT Facility

Explanation: An outbound LAT connection was attempted when outbound connections are disabled.

User Action: Enable connections using the LATCP command SET NODE. Refer to the *VMS LAT Control Program (LATCP) Manual* for more information.

STACKOVF, attempted stack overflow was detected

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A thread overflowed its stack.

User Action: Create the erring thread with a larger stack or redesign the code to require less stack space; for example, nest your calls less deeply or allocate less storage on the stack.

STARTUP, VMS startup begun at 'dd-mmm-yyyy hh:mm:ss.ss'

Facility: STDRV, System Startup Driver

Explanation: VMS has begun executing the system startup driver, which is used to start up individual VMS system processes and to start VMS after a reboot.

User Action: None. This is an informational message.

STKNOTCHANGE, the stock associated with a form cannot be changed

Facility: JBC, Job Controller

Explanation: A DEFINE/FORM command for an existing form specified /STOCK with a new stock value while references to the form are still outstanding.

User Action: Use the SHOW QUEUE/FULL command to locate existing references. Remove any outstanding references and reenter the DEFINE /FORM/STOCK request.

STRTOOLNG, string argument is too long - shorten

Facility: CLI, Command Language Interpreter (DCL)

Explanation: The specified string argument is too long.

User Action: Check the description of the command in the *VMS DCL Dictionary* for restrictions on the argument length. The *VMS DCL Concepts Manual* also describes the maximum allowable length of an argument for any command. Modify the string accordingly and retry the command.

VMS System Messages

SYMDEL, unexpected symbiont process termination

Facility: JBC, Job Controller and QMAN, Queue Manager

Explanation: A symbiont process exited without being requested to do so. The accompanying message provides additional information.

User Action: Take action based on the accompanying message. A process dump might have been created. This message can result from an unplanned node or cluster shutdown.

SYSBOOT-I-GBLPAGES have been trimmed

Facility: SYSBOOT, System Bootstrap Facility

Explanation: The combined size of the system and global page tables exceeds the VMS architectural maximum (4,194,303 pages). SYSBOOT has reduced the size of the global page table by decreasing the SYSGEN parameter GBLPAGES.

User Action: Review the ACTIVE value of the GBLPAGES parameter to make sure it is large enough to support normal system operation in your environment. Using SYSMAN, reevaluate the values of the parameters that determine the size of the system and global page tables, especially if the value computed by AUTOGEN has been overridden in MODPARAMS.DAT. (Refer to the *VMS SYSMAN Utility Manual*.)

SYSFAIL, system failed during execution

Facility: JBC, Job Controller

Explanation: The system crashed during execution of a batch or symbiont process.

User Action: Resubmit the batch job or restart the output queues previously associated with the affected symbiont process.

TIMED_OUT, timed condition wait expired

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: On a `cma_cond_timed_wait` routine, the timer expired before the condition was signaled or broadcast.

User Action: Take appropriate action based on program dependencies for the specific condition variable wait that timed out.

TIMEOUT, no response within timeout period

Facility: LAT, LAT Facility

Explanation: A LAT connection was lost because the remote node did not respond within the timeout period.

User Action: Check the network availability of the remote node. If this error persists, you may need to increase the retransmit limit on the local node.

TIMEOUT, transaction exceeded execution time limit from \$START_TRANS service

Facility: DDTM, DECdtm Services

Explanation: The transaction aborted because the time specified in the `timout` argument when calling \$START_TRANS has been exceeded.

User Action: None.

TMSCPLDERR, TMSCP server must be loaded using SYSGEN parameter TMSCP_LOAD

Facility: SYSGEN, System Generation Utility

Explanation: You attempted to load the TMSCP server using the TMSCP command within SYSGEN.

User Action: The proper way to load the TMSCP server is to set the SYSGEN parameter TMSCP_LOAD to 1. This action loads the server, which services all locally connected MSCP-type tape drives during SYSBOOT.

TODISCON, type ^'character' to disconnect the session

Facility: LAT, LAT Facility

Explanation: Use the specified control character to disconnect the SET HOST/LAT session.

User Action: Enter the specified control character.

TOOMANYSUB, SPAWN failed due to too many subprocesses; DIRECT mode used

Facility: STDRV, System Startup Driver

Explanation: The startup database directed the startup driver to run too many spawned subprocesses. STDRV ran one or more of the processes in the main startup procedure using DIRECT mode. System startup should complete normally.

User Action: You can use the SYSMAN utility's STARTUP commands to display and modify the startup database to spawn fewer subprocesses.

TOOMUCHINFO, size of data in request exceeds system constraints

Facility: JBC, Job Controller

Explanation: The amount of data specified for a record within the queue manager's database is too large.

User Action: Submit a Software Performance Report (SPR) to notify VMS Engineering that current constraints do not meet your needs.

TPSFAOERR, could not format display line

Facility: SDA, System Dump Analyzer

Explanation: The structure displayed contains data that could not be formatted properly.

User Action: If further analysis is required, use the SDA FORMAT command to examine the structure.

TPSINVBLK, invalid block type in specified block

Facility: SDA, System Dump Analyzer

Explanation: An attempt was made to copy an unrecognized structure.

User Action: For an active system, retry the command. For a system crash dump, submit a Software Performance Report (SPR) that describes the conditions leading to the error; include a BACKUP save set containing the output of the SDA command.

TPSTERM, TP Services structure display terminated prematurely

Facility: SDA, System Dump Analyzer

Explanation: The selected SDA command was unable to complete. The Transaction Processing (TP) structures displayed by this command are corrupt.

User Action: None.

TPSUTCERR, no valid timestamp

Facility: SDA, System Dump Analyzer

Explanation: The displayed structure can contain an optional timestamp. The structure displayed currently does not have a timestamp.

User Action: None. This is an informational message.

TSRVALLOAD, the TMSCP server is already loaded

Facility: SYSGEN, System Generation Utility

Explanation: You attempted to load the TMSCP server using the TMSCP command within SYSGEN. The TMSCP server has already been loaded in the recommended way.

User Action: The proper way to load the TMSCP server is to set the SYSGEN parameter TMSCP_LOAD to 1. This action loads the server, which services all locally connected MSCP-type tape drives during SYSBOOT.

UNDEFLINK, undefined link

Facility: LAT, LAT Facility

Explanation: You specified a link that does not exist on the local node.

User Action: Use the LATCP command SHOW LINK to see which links exist on the local node.

UNIMP, the specified DECthreads feature is not implemented

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: You attempted to use a feature that is not implemented in the version of DECthreads that you are running. This error can occur when a program developed on a system running a new version of DECthreads is executed on a system that is running an old version of DECthreads.

User Action: Use a later version of DECthreads that supports the feature or do not attempt to use the feature with an old version of DECthreads.

UNINITEXC, uninitialized exception raised

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: The EXC_HANDLING.H package, which provides portable exceptions for the C language, has attempted to raise an exception that has not been initialized.

User Action: Study the error messages to determine the program location where the uninitialized exception is being raised. Use the **exception_init** macro defined in the EXC_HANDLING.H package to initialize the exception.

UNREACHABLE, node 'node-name' not reachable

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt to a specified node and service failed because the node offering the service is unreachable.

User Action: Locate the node and determine what is preventing connections from occurring.

USE_ERROR, requested operation is inappropriate for the specified object

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: The state or type of an object is inappropriate for the operation; for example, the operation attempts to unlock a mutex that is not locked.

User Action: Determine which routine caused the error and consult the documentation to learn which object states are appropriate for the routine.

VA_IN_USE, virtual address already in use

Facility: SYSTEM, VMS System Services

Explanation: A PTD\$CREATE request specified a buffer address that is already being used by another PTD\$CREATE request or by another system memory management facility such as SYS\$CRMPSC.

This message can occur when the main image or a sharable image is based. An image is based if a linker options file is used to specify a base virtual address at which the image should be loaded or if certain language constructs are used that produce nonrelocatable code.

User Action: Allocate a new region of virtual memory to be used for I/O buffers, then reissue the PTD\$CREATE request specifying the new region.

VCLIM, LAT circuit limit reached

Facility: LAT, LAT Facility

Explanation: The maximum number of allowable LAT circuits has been reached.

User Action: Retry the operation when a circuit becomes available.

VCSESLIM, session limit for LAT circuit reached

Facility: LAT, LAT Facility

Explanation: A LAT connection attempt failed because the connection between the local node and the destination node already has the maximum number of sessions allowed.

User Action: Attempt a connection to another node offering the same service or wait until a session becomes available.

VETOED, participant vetoed commitment

Facility: DDTM, DECdtm Services

Explanation: The transaction aborted because a resource manager could not commit the transaction.

User Action: Determine why the resource manager could not commit the transaction and correct the error.

VMS System Messages

WORKING, customer defined text

Facility: LAVC, Local Area VAXcluster

Explanation: The local area VAXcluster network failure analysis has determined that this network component is working.

User Action: If the network component is indeed working, no user action is required.

However, if this message displays when the network component is not working, have the system manager or network manager look into the network problem. In such a case, the network description does not accurately represent the physical network. Review the defined network components by calling SYS\$LAVC_DEFINE_NET_COMPONENT. Review the defined network path descriptions by calling SYS\$LAVC_DEFINE_NET_PATH. Correct any problems as necessary.

WRONGMUTEX, wrong mutex specified in condition wait

Facility: CMA, DECthreads (Multithreading Run-Time Library)

Explanation: A thread attempted to wait for a condition variable that already has at least one thread waiting, and that thread has specified a different mutex. DECthreads requires that all threads concurrently waiting for a condition variable specify the same mutex.

User Action: Design your code so that each condition variable represents a particular state of shared data that is protected by a given mutex.

WRONGSTATE, invalid transaction state for requested event

Facility: SYSTEM, VMS System Services

Explanation: The transaction is in the wrong state for the attempted operation.

User Action: If this message is returned by the \$ABORT_TRANS or \$END_TRANS service, correct the error in the program. Otherwise, submit a Software Performance Report (SPR) that describes the conditions leading to the error. Include a BACKUP save set containing the output of the LMCP DUMP command for the local transaction log file and the output of the DCL DUMP command for the same log file.

ZEROLINK, zeroed counters for link 'link-name'

Facility: LAT, LAT Facility

Explanation: LATCP has reset the counters for the specified link to zero.

User Action: None.

ZERONODE, zeroed counters for node 'node-name'

Facility: LAT, LAT Facility

Explanation: LATCP has reset the counters for the specified node to zero.

User Action: None.

ZEROSERVICE, zeroed counters for service 'service-name'

Facility: LAT, LAT Facility

Explanation: LATCP has reset the counters for the specified service to zero.

User Action: None.

Part III

System Management Features

This part contains the following chapters:

- Chapter 5, VMS Batch and Print Queuing System
- Chapter 6, LADCP Utility
- Chapter 7, Clusterwide Tape Serving
- Chapter 8, VMS Volume Shadowing Phase II Enhancements
- Chapter 9, LAT New Features
- Chapter 10, VMS License Management Facility
- Chapter 11, Movefile Command Qualifiers

Part III

System Management Features

- 1. The first feature is the ability to manage the system's configuration.
- 2. The second feature is the ability to manage the system's security.
- 3. The third feature is the ability to manage the system's performance.
- 4. The fourth feature is the ability to manage the system's resources.
- 5. The fifth feature is the ability to manage the system's logs.
- 6. The sixth feature is the ability to manage the system's updates.
- 7. The seventh feature is the ability to manage the system's backups.
- 8. The eighth feature is the ability to manage the system's monitoring.
- 9. The ninth feature is the ability to manage the system's alerts.
- 10. The tenth feature is the ability to manage the system's reports.

VMS Batch and Print Queuing System

This chapter contains system management information about the new VMS batch and print queuing system. For information about setting up and managing a queuing system, see the *Guide to Maintaining a VMS System*.

Note

Digital recommends that you take advantage of the new features in the batch and print queuing system. However, if you cannot do so at this time, your queuing system will continue to work with VMS Version 5.4 queue commands, with the following exception: If no queue database exists, you must specify the `/NEW_VERSION` qualifier with the `START/QUEUE/MANAGER` command to create a queue database.

5.1 Clusterwide Queue Manager

In the previous batch and print queuing system, a queue manager ran on each node in a cluster, as part of the node's job controller process. Each node's job controller/queue manager accessed a distributed queue database to control queuing operations. User processes, symbionts, and batch jobs communicated with the queue manager through their local job controller. Figure 5-1 illustrates the queue manager's role in the previous batch and print queuing system.

With the new VMS batch and print queuing system, queue manager and job controller functions are separate. A single queue manager process acts as a clusterwide server, accessing the queue database for all processes in a cluster. Job controllers, user processes, and symbionts on each node communicate directly with the centralized queue manager through a shared interprocess communications (IPC) interface link. An IPC is an internal VMS communications mechanism. Figure 5-2 illustrates the role of the new clusterwide queue manager.

The new centralized design reduces disk activity associated with the distributed design. It also enables the queue manager to fail over to another node if the node on which it is running leaves the cluster.

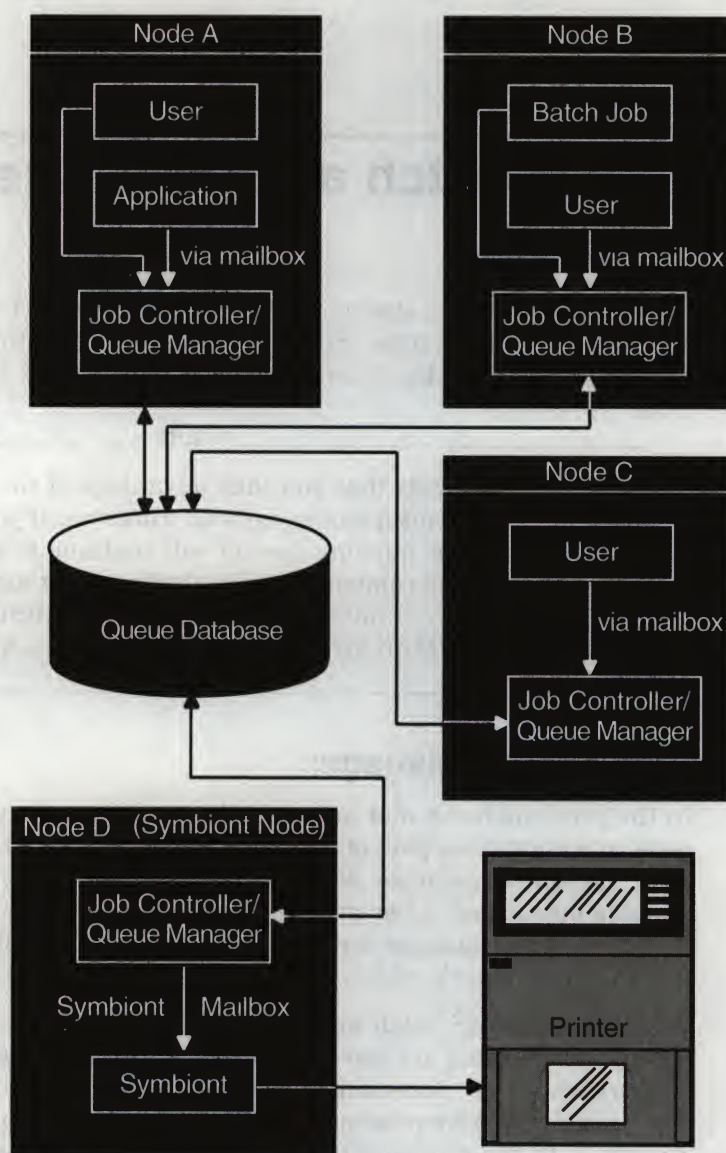
With the new queuing system, the queue manager handles all queuing requests. The job controller performs all other activities, including:

- Creating and monitoring batch, symbiont, and queue manager processes
- Processing the DCL command `START/QUEUE/MANAGER`

VMS Batch and Print Queuing System

5.1 Clusterwide Queue Manager

Figure 5-1 VMS Version 5.0 Queue Manager



ZK-3522A

- Handling queue manager failover

The changes to the queue manager affect those commands used to start and stop the queue manager. For more information, see Section 5.3.

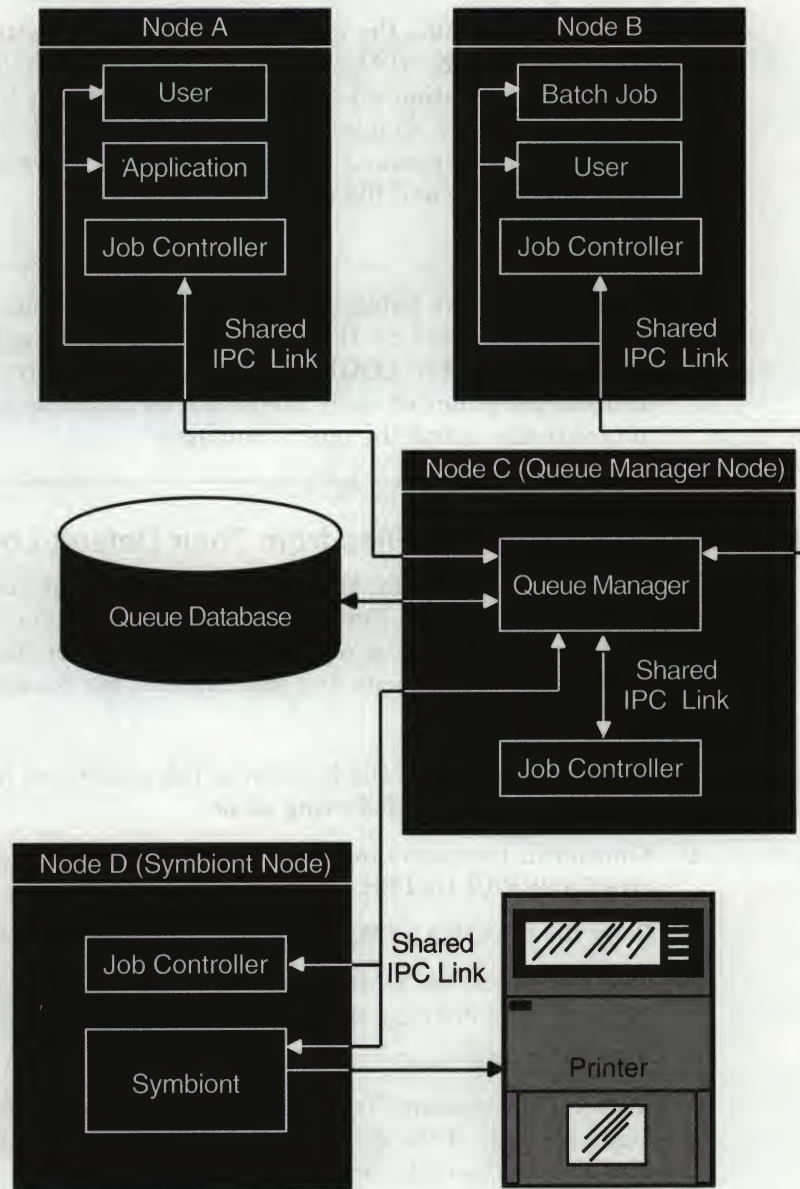
5.2 New Queue Database Design

The new VMS batch and print queuing system includes a new queue database. The file previously used as the queue database, JBCSYSQUE.DAT, is no longer used.

The new queue database consists of the following new files:

QMAN\$MASTER.DAT, the master file

Figure 5-2 VMS Version 5.5 Queue Manager



ZK-3521A -GE

SYS\$QUEUE_MANAGER.QMAN\$QUEUES, the queue file
SYS\$QUEUE_MANAGER.QMAN\$JOURNAL, the journal file

SYS\$COMMON:[SYSEXE] is the default location for all three queue database files. However, you can move the files to another location. For more information, see Section 5.2.1.

The master file contains the following information:

- The location of the queue and journal files
- Definitions of forms and characteristics
- A list of queue names

VMS Batch and Print Queuing System

5.2 New Queue Database Design

- A list of nodes allowed to run the queue manager

The queue file contains the queue definitions you create when you enter INITIALIZE/QUEUE, START/QUEUE, and SET QUEUE commands. The journal file contains information allowing the queue manager to return to the last known state should a standalone machine be stopped unexpectedly or should a VAXcluster member running the queue manager leave the cluster. The journal file also contains job and file record definitions.

Note

The disk or disks holding the three queue database files should be mounted by the startup command procedure SYS\$MANAGER:SYLOGICALS.COM. It is important that the disk or disks are mounted early, so the queue database is available before the job controller starts the queue manager.

5.2.1 Moving Queue Database Files from Their Default Location

The upgrade procedure for the VMS batch and print queuing system gives you the opportunity to move the queue database files from their default locations during the upgrade. If you need to move the master file, see Section 5.2.1.1. If you need to move the queue and journal files, see Section 5.2.1.2.

5.2.1.1 Moving the Master File

The master file contains the location of the queue and journal files. To move the master file, perform the following steps:

1. Shut down the queue manager by entering the DCL command STOP/QUEUE /MANAGER/CLUSTER.
2. Copy the file QMAN\$MASTER.DAT to a new location.
3. Edit the file SYS\$COMMON:[SYSMGR]SYLOGICALS.COM to add the following line defining the logical name QMAN\$MASTER:

```
$ DEFINE/SYSTEM/EXECUTIVE_MODE QMAN$MASTER directory-name
```

where *directory-name* is the directory specification for the directory where the file is located. If the directory is on a disk other than the default, you must also specify the disk name.

In a VAXcluster environment, QMAN\$MASTER must be identically defined on all nodes in the cluster.

4. Restart the queue manager with the DCL command START/QUEUE /MANAGER.

5.2.1.2 Moving the Queue and Journal Files

The queue and journal files are not required to reside in the same directory as the master file; however, if you move the queue and journal files, they must reside together in the same directory. The master file contains the location of these files.

To move the queue file (SYS\$QUEUE_MANAGER.QMAN\$QUEUES) and the journal file (SYS\$QUEUE_MANAGER.QMAN\$JOURNAL) to a new location, perform the following steps:

1. Shut down the queue manager by entering the DCL command STOP/QUEUE/MANAGER/CLUSTER.

2. Copy the queue and journal files to their new location. They must reside together in the same directory.
3. Restart the queue manager by entering the following DCL command:

```
$ START/QUEUE/MANAGER SYSMAN$DISK:[MY_QUE_JOU_DIR]
```

where SYSMAN\$DISK:[MY_QUE_JOU_DIR] is the specification for the directory containing the queue and journal files.

Note

In a VAXcluster environment, if the string substituted for *directory-name* in the START/QUEUE/MANAGER command is a concealed logical name, it must be identically defined on all nodes in the cluster.

Once you enter the START/QUEUE/MANAGER command, the directory location you enter is stored in the queue database. If you need to restart the queue manager, you do not need to respecify the directory location.

5.3 Starting and Stopping the Queue Manager

Changes in the new VMS batch and print queuing system affect the commands used to start and stop the queue manager. The following sections describe these changes.

5.3.1 Starting the Queue Manager

In the previous queuing system, the START/QUEUE/MANAGER command started a queue manager process that provided queuing services only for the node on which the command was entered. A queue manager process ran on each node from which the START/QUEUE/MANAGER command was entered.

With the new VMS batch and print queuing system, START/QUEUE/MANAGER is a clusterwide command. It starts up a single queue manager process that provides queuing services for all nodes in a VAXcluster system.

To start the clusterwide queue manager, enter the following command:

```
$ START/QUEUE/MANAGER
```

If no queue database exists, specify the /NEW_VERSION qualifier to create the queue database.

Caution

If you specify the /NEW_VERSION qualifier and you already have a queue database, the START/QUEUE/MANAGER command deletes certain information from the existing database. Do not use the /NEW_VERSION qualifier with the START/QUEUE/MANAGER command unless no database exists, or you no longer need the existing database.

If you want to place your queue and journal files in a location other than the default location of SYS\$COMMON:[SYSEXEC], you must specify the location with the START/QUEUE/MANAGER command when you start the queue manager. For instructions on moving queue and database files, see Section 5.2.1.2.

VMS Batch and Print Queuing System

5.3 Starting and Stopping the Queue Manager

5.3.1.1 Customizing Queue Manager Failover

In a VAXcluster environment, the new queue manager will automatically fail over to another node if the node on which it is running leaves the cluster. However, you can use the /ON qualifier to specify the order in which the nodes claim the queue manager during failover and, if desired, you can use the /ON qualifier to limit the nodes that run the queue manager. Use the following command syntax:

```
START/QUEUE/MANAGER/ON=(node-list)
```

Digital recommends that you specify the asterisk wildcard (*) as the last node in the node list to indicate that any remaining unlisted node can claim the queue manager, with no preferred order. If you want to exclude certain nodes from being eligible to run the queue manager, you also cannot use the asterisk wildcard. You cannot specify the asterisk wildcard as part of a node name.

In the following example, the queue manager will be started on node ALPHA (if ALPHA is available):

```
$ START/QUEUE/MANAGER/ON=(ALPHA,BETA,GAMMA,*)
```

If node ALPHA exits the cluster, node BETA will start up the queue manager process (if BETA is available). During the transition from ALPHA to BETA, queues on BETA and GAMMA are not stopped. All requests to the queuing system—for example, PRINT, SUBMIT, and SHOW ENTRY requests—will complete as expected. If ALPHA, BETA, and GAMMA are not available, any remaining node can claim the queue manager.

To change a list of nodes on which the queue manager can run, reenter the START/QUEUE/MANAGER command with the new node list. The new command is stored in the queue database, and the queue manager is stopped and restarted with the new node list. For more information, see Section 5.3.1.3.

5.3.1.2 Automatic Queue Manager Restart

When you enter the START/QUEUE/MANAGER command, it is stored in the queue database. Thereafter, the job controller automatically restarts the queue manager during reboot unless a STOP/QUEUE/MANAGER/CLUSTER command has been entered.

You do not need to include the START/QUEUE/MANAGER command in your site-specific startup procedure. The START/QUEUE/MANAGER command is no longer included in the startup procedure template SYSTARTUP_V5.TEMPLATE.

5.3.1.3 If the Queue Manager Is Already Started

If the queue manager is already running and you enter the START/QUEUE/MANAGER command with qualifier values different from those used to start the queue manager initially, the queue manager will be changed to reflect the new qualifier values.

If the queue manager is running and this command is entered with no new qualifier values, the job controller will check to see if one or more preferred queue manager nodes is stored in the queue database. See Section 5.3.1.1 for information on how to specify preferred queue manager nodes using the /ON qualifier with the START/QUEUE/MANAGER command.

If you specify one or more preferred nodes, and the queue manager is running on a node other than the first available specified node, the job controller attempts to restart the queue manager on the first available preferred node. Despite the transition, queues on running nodes are not stopped and all requests to the queuing system will complete as expected.

5.3.1.4 Obsolete Qualifiers

The /EXTEND, /BUFFER_COUNT, and /RESTART qualifiers to the DCL command START/QUEUE/MANAGER are obsolete with the new batch and print queuing system.

5.3.2 Stopping the Queue Manager

To stop the clusterwide queue manager, enter the following command:

```
$ STOP/QUEUE/MANAGER/CLUSTER
```

This command stops the queue manager process. The process remains stopped until the DCL command START/QUEUE/MANAGER is entered. Cluster transitions will not change the state of the queue manager. Newly available nodes will not attempt to start the queue manager (unless the START/QUEUE/MANAGER command is executed).

5.3.3 Stopping Queues on a Node

To stop all queues on a node, enter the following command:

```
$ STOP/QUEUES/ON_NODE
```

By default, this command stops all queues on the node from which the command is entered. To stop queues on another node, specify the node name with the /ON_NODE qualifier as follows:

```
$ STOP/QUEUES/ON_NODE=BETA
```

This command implicitly disables the autostart feature on the node on which the command takes effect. As a result, queues started with an autostart list fail over to the next available node in that list that has autostart enabled. For information about the autostart feature, see Section 5.4.

The STOP/QUEUES/ON_NODE command replaces the DCL command STOP/QUEUE/MANAGER. In previous versions, STOP/QUEUE/MANAGER stopped the queue manager on a single node in a cluster. Because the queue manager is now clusterwide and not node specific, the STOP/QUEUE/MANAGER command is obsolete. If you enter the command STOP/QUEUE/MANAGER, it will perform the same function as the new DCL command STOP/QUEUES/ON_NODE.

5.4 The Autostart Feature

The autostart feature simplifies startup and ensures high availability of execution queues in a cluster. An autostart queue is a special type of execution queue that makes use of the autostart feature. The autostart feature lets you do the following:

- Start all autostart queues on a node with a single command
- Specify a list of nodes (within a VAXcluster environment) to which a queue can automatically fail over if a node is removed from the cluster

For these reasons, Digital recommends that you use autostart queues whenever possible.

The following DCL commands are new or changed to support the autostart feature:

- INITIALIZE/QUEUE/AUTOSTART_ON=(node::[device] [...])
- ENABLE AUTOSTART[/QUEUES][ON_NODE=node-name]

VMS Batch and Print Queuing System

5.4 The Autostart Feature

- START/QUEUE/AUTOSTART_ON=(node::[device] [...])
- DISABLE AUTOSTART[/QUEUES]/[ON_NODE=node-name]

The following sections discuss these commands in more detail.

5.4.1 Designating a Queue as an Autostart Queue

To designate a queue as an autostart queue, specify one of the following DCL commands:

- INITIALIZE/QUEUE/AUTOSTART_ON=node::[device]
- START/QUEUE/AUTOSTART_ON=node::[device]

Both *node* and *device* must be specified for output queues, for example, GOOD::MYPRINTER. For batch queues, only *node* is required, for example, GOOD::.

You cannot specify the /AUTOSTART_ON=node::[device] qualifier with the /GENERIC qualifier or the /ON=node::[device] qualifier.

Caution

The node name you specify as *node* is not checked to determine if it is an existing node name. Be sure to specify a correct node name.

5.4.1.1 Setting Up Autostart Queues for Automatic Failover

To increase the availability of execution queues in a cluster, you can set up an autostart queue to execute on one of several nodes in a list. If the node on which an autostart queue is running leaves the cluster, the queue will automatically fail over to the next available node in the list on which autostart is enabled.

To specify the list of nodes to which an autostart queue can fail over, include the list with the /AUTOSTART_ON qualifier for the INITIALIZE/QUEUE or START/QUEUE command as follows:

```
INITIALIZE/QUEUE/AUTOSTART_ON=(node::[device] [...]) queue
START/QUEUE/AUTOSTART_ON=(node::[device] [...]) queue
```

Caution

The node name you specify as *node* is not checked to determine if it is an existing node name. Be sure to specify a correct node name.

For example:

```
$ INITIALIZE/QUEUE/AUTOSTART_ON=(DATA::FOO,WARF::BAR,DEANNA::DOO) MYQUEUE
$ START/QUEUE MYQUEUE
```

In this example, the output queue named MYQUEUE will start on the first node in the list for which the ENABLE AUTOSTART command is entered. If the node on which MYQUEUE is executing is taken out of the cluster, the queue will be stopped on that node and will fail over to the first available node in the list on which autostart has been enabled. The queue manager will automatically restart the queue on the new node.

As long as one of the three nodes is running with autostart enabled, this queue will be started and available to execute print jobs. If all three nodes in the example have been shut down, the queue will remain stopped until one of the three nodes joins the cluster and executes the ENABLE AUTOSTART command.

5.4.2 Enabling Autostart on a Node

The command ENABLE AUTOSTART/QUEUES notifies the queue manager to automatically restart all active autostart queues on a system. It also notifies the queue manager to automatically start any active autostart queue that fails over to the system. An autostart queue is active if it has been started initially and has not been stopped with the STOP/QUEUE/NEXT or STOP/QUEUE/RESET command. By default, the command affects the node from which it is entered. However, you can specify the /ON_NODE=nodename qualifier to enable autostart on a different node. For example:

```
$ ENABLE AUTOSTART/QUEUES/ON_NODE=NODEX
```

The /QUEUES qualifier is optional.

When a node reboots, autostart is disabled until you enter the ENABLE AUTOSTART/QUEUES command. Add this command to your system startup procedures following the commands that configure printer devices and mount important disks. The ENABLE AUTOSTART/QUEUES command is included in the template startup procedure SYSTARTUP_V5.TEMPLATE provided with VMS Version 5.5. Use this command in your startup procedure instead of separate START/QUEUE commands to restart each autostart execution queue.

Non-autostart execution queues (those created or started with the /ON=node::[device] qualifier) will not be automatically restarted when a node reboots and therefore must be restarted with the START/QUEUE command.

5.4.3 Starting Autostart Queues

You must start an autostart queue initially, in one of the following ways:

- Specify the /START qualifier in the INITIALIZE/QUEUE command used to create the queue.
- Enter a START/QUEUE command after you create the queue.

Autostart must be enabled on the node as explained in Section 5.4.2 for the queue to begin executing jobs. Once autostart is enabled and the queue is started initially, the queue will remain started until either of the following occurs:

- Autostart is disabled on the node with the DISABLE AUTOSTART or STOP /QUEUES/ON_NODE command or if the node leaves the cluster.
- The queue is stopped with a STOP/QUEUE/NEXT or STOP/QUEUE/RESET command.

5.4.4 Preventing Autostart Queues from Starting

With autostart queues, the STOP/QUEUE/NEXT or STOP/QUEUE/RESET command stops a queue and marks it inactive for autostart until the START /QUEUE command is entered. The STOP/QUEUE/NEXT or STOP/QUEUE /RESET command prevents an autostart queue from being automatically restarted.

You might use this feature to prevent an autostart output queue from accidentally restarting when a printer is being serviced.

5.4.5 Disabling Autostart on a Node

The `DISABLE AUTOSTART/QUEUES` command notifies the queue manager to perform the following tasks on the affected node:

- Prevent autostart queues from failing over to the node.
- Mark all autostart queues on the node as “stop pending” in preparation for a planned shutdown. This lets jobs currently executing on the queues complete.
- Upon completion of any jobs currently executing on one of the node’s autostart queues, force the queue to fail over to the next available node in the queue’s failover list on which autostart is enabled. (An autostart queue can fail over only if you have set it up to run on more than one node, as specified in Section 5.4.1.1.)

By default, the command affects the node from which it is entered. However, you can specify the `/ON_NODE=node` qualifier to disable autostart on another node. The `/QUEUES` qualifier is optional.

The `DISABLE AUTOSTART/QUEUES` command has been added to the shutdown command procedure `SHUTDOWN.COM` and will be automatically executed when you shut down a node using `SHUTDOWN.COM`. If you shut down a node without using `SHUTDOWN.COM` and the node is running autostart queues, you might want to enter the `DISABLE AUTOSTART` command before shutting down the node.

The `DISABLE AUTOSTART` command affects autostart queues only. You must still stop all non-autostart queues executing on the node by entering one of the following commands:

```
STOP/QUEUE/RESET  
STOP/QUEUE/NEXT  
STOP/QUEUES/ON_NODE
```

In addition to the changes described in this section, the following VMS components have been changed to support the autostart feature:

- `F$GETQUI` lexical function (see Section 3.6)
- `SYS$SNDJBC` and `SYS$GETQUI` system services (see *VMS System Services Reference Manual*)
- `LIB$GETQUI` run-time library routine (see Section 13.1)

LADCP Utility

The LAD control program (LADCP) is the utility program that you use to configure and control the local area disk (LAD) protocol on VMS host systems. VMS systems that use LAD services are called LAD client nodes.

You can use LADCP to do the following:

- Establish **bindings** to LAD services, which creates a new DADn: virtual disk unit on the local VAX system
- Remove bindings to LAD services

You can control service access by using a service access password. You can also write-protect LAD services. In this case, local VMS users of the DADn: device unit receive an error if they attempt a write operation to the unit.

The LAD protocol allows you to access disk media that reside on a Digital InfoServer system as though they were locally connected to your VAX system. This allows several VMS client nodes to share the same disk media, eliminating the need for duplicate disk drives and media.

For more information about the LADCP utility, refer to *VMS LAD Control Program (LADCP) Manual*.

Clusterwide Tape Serving

Included in VMS Version 5.5 is the VMS tape mass storage control protocol (TMSCP) server. The tape server allows the system manager to make locally connected tape drives cluster-accessible tapes. A cluster-accessible tape is a tape that every node in the cluster can recognize and access.

The tape server allows nodes without a locally connected tape drive to gain direct access to a tape drive connected to another node.

Once the server has been loaded and tape devices have been set as served, the devices can be accessed from any node in the VAXcluster using DCL commands. INITIALIZE, MOUNT, and BACKUP operations can be done on remote tape devices in the same way as they are currently done using locally connected devices.

Note

Tape drives are not shared devices. Only one user can access a tape at a time. With the tape server, served tape drives are accessible to all nodes in a cluster, but can be allocated and accessed by only one process at a time.

See the *VMS VAXcluster Manual* for details about implementing the TMSCP server.

7.1 Loading the Magnetic Tape Server

By default, VMS does not load the tape server software. To implement the server, the system manager must modify the SYSGEN parameter TMSCP_LOAD and, optionally, the TAPE_ALLOCLASS parameter.

7.1.1 TMSCP_LOAD Parameter

A new SYSGEN parameter, TMSCP_LOAD, has been created to allow for the loading of the TMSCP server software. The TMSCP_LOAD parameter also sets locally connected tapes as served.

When TMSCP_LOAD is set to zero, it inhibits the loading of the tape server and the serving of local tapes. When TMSCP is set to 1, it loads the tape server into memory at the time the system is booted and makes all directly connected tape drives available clusterwide. The following table describes the two states of the TMSCP_LOAD parameter:

Clusterwide Tape Serving

7.1 Loading the Magnetic Tape Server

State	Function
0	Do not load the TMSCP tape server. Do not serve any local tape devices clusterwide.
1	Load the TMSCP tape server. Serve all local TMSCP tape devices clusterwide.

The parameter has the following restriction for VMS Version 5.5:

- The TMSCP tape server will serve TMSCP tape drives only.

DSSI tapes (for example, the TF85) are TMSCP tape devices. Tape devices connected to an HSC are also TMSCP devices. SCSI tapes such as the TL and TZ tape devices (which are displayed as MK cn) are not TMSCP devices.

Some tapes can be TMSCP devices depending on their configuration. A TK50 in a MicroVAX system or being used as the console media for a VAX 6000-series computer is a TMSCP device. A TK50 on a VAXstation 2000 computer is not a TMSCP tape device.

TMSCP devices include the TA79, TA81, TA90, TA90E, TA91, TF70, TF85, TF737, TF857, TK50, TK70, TU81, TU81+, RV20, and RV60.

You can use the SHOW DEVICE command to identify TMSCP tape devices. Use the SHOW DEVICE M command to obtain a list of tape (and mailbox) devices. MU and MI tape devices are TMSCP tapes, so if SHOW DEVICE M displays a TU81 tape device as MUA0, the device is a TMSCP device.

Note

In VMS Version 5.5, the DCL command SHOW DEVICE/SERVED does not display the names of served tapes.

SDA (the System Dump Analyzer Utility) can also be used to determine if a tape device is a TMSCP tape. To invoke SDA, enter ANALYZE/SYSTEM from a privileged account at the DCL prompt. Then use the SDA command SHOW DEVICE MU cn , where c is the controller letter and n is the device unit number. The display will be similar to the following:

```

SAMPL$MUB6          TK70          UCB address: 80C00BB0
Device status:      00000010 online
Characteristics:    0C444038 dir,sdi,sqd,fod,avl,elg,idv,odv
                   000022A1 clu,mscp,srv,nm,loc

Owner UIC [000000,000000]  Operation count      0  ORB address      80C00CD0
PID          00000000      Error count          0  DDB address      81C17600
Alloc. lock ID 00000000      Reference count      0  DDT address      80B6D904
Alloc. class    102          BOFF                  0000  CRB address      81C17580
Class/Type      02/0F          Byte count          0000  PDT address      80B6CFA0
Def. buf. size  2048          SVAPTE              00000000  CDD address      80B71930
DEVDEPEND       000004C0      DEVSTS              0000  I/O wait queue   empty
DEVDEPN2        00000008      RWAITCNT            0000
FLCK index      34          Object count        0
DLCK address    00000000

```

In the second line listing the characteristics, the symbol "mscp" indicates that the device, a TK70, is a TMSCP device and the symbol "srv" indicates that this device is currently served to all the VAXcluster nodes.

7.1.2 TAPE_ALLOCLASS Parameter

To serve tapes, the SYSGEN parameter TMSCP_LOAD must be set to 1. Additionally, the SYSGEN tape allocation class parameter, TAPE_ALLOCLASS, must follow the same rules as the SYSGEN parameter ALLOCLASS does for serving disks. These rules are

- VAX or HSC nodes connected to a dual-path tape must have the same nonzero tape allocation class value.
- All cluster-accessible tapes on nodes with a nonzero allocation class value must have unique names. For example, if two VAX nodes in a VAXcluster have the same tape allocation class value, it is invalid for both nodes to have a tape named MUA0. This restriction also applies to HSCs.
- Single-ported tapes with an allocation class value of zero can have the same unit number on different cluster nodes.

Note that zero is the default tape allocation class value. In a mixed-interconnect cluster, all of the following must have a nonzero tape allocation class value:

- HSCs
- Systems serving HSC tapes
- Systems connected to dual-path tapes

VMS Volume Shadowing Phase II Enhancements

VMS Volume Shadowing Phase II supports a new SYSGEN parameter, SHADOW_MBR_TMO, which lets you specify the timeout period for recovering a shadow set member before it is removed from a shadow set. Previously, you used the SYSGEN parameter VMDS3 to specify the number of seconds before timing out.

Volume Shadowing Phase II now also provides support for SCSI devices.

Sections 8.1 and 8.2 describe these new features in more detail. Note that VAX Volume Shadowing (Phase I) does not include support for either of these features.

8.1 Specifying the Shadow Set Member Recovery Timeout Period

You can set the SHADOW_MBR_TMO parameter to specify the number of seconds (from 1 to 65,535 decimal) during which recovery of a repairable shadow set is attempted. If you do not specify a number, VMS uses the default value of 20 seconds.

The following example shows how to set the value of SHADOW_MBR_TMO to 10 seconds:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> USE CURRENT
SYSGEN> SET SHADOW_MBR_TMO 10
SYSGEN> SHOW SHADOW_MBR_TMO
```

Parameter Name	Current	Default	Min.	Max.	Unit	Dynamic
SHADOW_MBR_TMO	10	20	0	65538	SECS	D

```
SYSGEN> WRITE CURRENT (or WRITE ACTIVE)
SYSGEN> EXIT
$
```

Because SHADOW_MBR_TMO is a dynamic parameter, you should use the SYSGEN command WRITE CURRENT to change its value permanently. To change temporarily the value of SHADOW_MBR_TMO on a running system, use the SYSGEN command WRITE ACTIVE.

Note

If there is currently a value in VMDS3 relevant to VMS Volume Shadowing Phase II, you can clear the value.

The SHADOW_MBR_TMO parameter is valid only for use with VMS Volume Shadowing (Phase II). You cannot set this parameter for use with VAX Volume Shadowing (Phase I).

VMS Volume Shadowing Phase II Enhancements

8.2 Volume Shadowing Phase II Supports Digital SCSI Devices

8.2 Volume Shadowing Phase II Supports Digital SCSI Devices

VMS Volume Shadowing (Phase II) now provides full support for all Digital Small Computer System Interface (SCSI) devices and for some other-vendor SCSI devices. VMS Volume Shadowing can support other-vendor devices that implement readl/writel commands because phase II shadowing software makes use of the optional SCSI readl (read long) and writel (write long) commands. Because VMS Volume Shadowing Phase II requires compatibility among the physical units in a shadow set, any supported SCSI device can be included in a phase II shadow set as long as its physical geometry is identical to the other SCSI devices in the shadow set. SCSI shadow set members can be located anywhere in a VAXcluster system.

Example 8-1 illustrates how you can use the SDA command SHOW DEVICE to determine whether or not a disk has readl/writel support. If the NOFE (No Forced Error) flag is set, the disk device does *not* have readl/writel commands implemented. In Example 8-1, the NOFE flag is shown at the end of the line following the line that begins with the word Characteristics. This flag indicates that the DKA200 device does not have forced error capability.

Example 8-1 Showing Device Characteristics Using the SDA SHOW DEVICE Command

```
SDA> SHOW DEV DKA200
BUBLA$DKA200          RZ23          UCB address: 803385B0

Device status:      00000010 online
Characteristics:    1C454008 dir,fod,shr,avl,elg,idv,odv,rnd
                   05000221 clu,mscp,nnm,scsi,nofe

Owner UIC [000000,000000]  Operation count      231  ORB address  803386E0
PID          00000000      Error count          0   DDB address  805655E0
Alloc. lock ID 00000000      Reference count      0   DDT address  8037B978
Alloc. class    5           Online count          0   CRB address  80527F70
Class/Type      01/31       BOFF                  0000  PDT address  8030C1A0
Def. buf. size   512        Byte count            0000  CDDb address 80338740
DEVDEPEND        03080821   SVAPTE              00000000  I/O wait queue empty
DEVDEPN2         00000000   DEVSTS              0004
FLCK index       34        RWAITCNT             0000
DLCK address     00000000
```

If you attempt to mount a SCSI device that does not have forced error capability into a shadow set, the MOUNT command fails and returns an informational status message. The following example shows the error message that results when you attempt to mount the DKA200 disk device:

```
$ MOUNT/SYS DSA101/SHAD=$5$DKA200: YELLOW
%MOUNT-I-DEVNOFE, device does not support FORCED ERROR handling.
```

You can mount SCSI devices that do not have forced error capability into phase II shadow sets using the /OVERRIDE=NO_FORCED_ERROR qualifier. This qualifier inhibits the protection checks performed by the MOUNT command. The following example shows how you use the /OVERRIDE=NO_FORCED_ERROR qualifier to mount the DKA200 disk device in a shadow set:

```
$ MOUNT/SYS/OVERRIDE=NO_FORCED_ERROR DSA101/SHAD=$5$DKA200: YELLOW
%MOUNT-I-MOUNTED, YELLOW          mounted on _DSA101:
%MOUNT-I-SHDWMEMSUCC, _$5$DKA200: (BUBLA) is now a valid member of the shadow set
```


VMS Volume Shadowing Phase II Enhancements

8.2 Volume Shadowing Phase II Supports Digital SCSI Devices

Note that a SCSI device mounted with the `/OVERRIDE=NO_FORCED_ERROR` qualifier will be dropped from the shadow set during a full copy operation if the device is the target of the operation and a bad block is encountered on the source device.

VAX Volume Shadowing (Phase I) does not include support for either Digital SCSI devices or other vendor's SCSI devices.

The following information is provided for the purpose of identifying the data sets that are included in the U.S. Volume 3: Building Phase II Superfund Digital Data. The information is provided for the purpose of identifying the data sets that are included in the U.S. Volume 3: Building Phase II Superfund Digital Data.

The following information is provided for the purpose of identifying the data sets that are included in the U.S. Volume 3: Building Phase II Superfund Digital Data. The information is provided for the purpose of identifying the data sets that are included in the U.S. Volume 3: Building Phase II Superfund Digital Data.

LAT New Features

The LAT software included in the VMS Version 5.5 operating system has been significantly enhanced. These changes affect the following operations:

- Starting up the LAT protocol software
- Using the site-specific LAT startup command procedure to customize LAT characteristics
- Using the new SET HOST/LAT command to establish outbound connections
- Using new commands and qualifiers with the LAT Control Program (LATCP)

Note

You can enter LATCP commands either at the LATCP> prompt or as a DCL command (interactively or in a program). If you choose the latter method, you must first define LCP and then precede each DCL command with that symbol, as shown in the following example:

```
$ LCP ::= $LATCP
$ LCP SET NODE/STATE=ON
```

- Using the new LAT ancillary control process (LATACP) to manage the services database
- Using the enhancements made to the QIO interface (described in Chapter 17)

This chapter includes complete information about starting, customizing, and managing the new LAT software. For additional information, see Chapter 17, the revised *VMS LAT Control Program (LATCP) Manual*, and the *VMS Version 5.5 Release Notes*.

9.1 Starting Up the LAT Protocol Software

To set up your node as a LAT service node and start the LAT protocol software on your system each time the system boots, edit SYS\$MANAGER:SYSTARTUP_V5.COM to add the following line:

```
$ @SYS$STARTUP:LAT$STARTUP.COM
```

When SYSTARTUP_V5.COM executes this command, it invokes LAT\$STARTUP.COM, which in turn invokes the LAT\$CONFIG and LAT\$SYSTARTUP command procedures.

You can append any of the following arguments to the command line that invokes LAT\$STARTUP to specify unique LAT characteristics for your node. The procedure will pass these arguments to LAT\$SYSTARTUP.COM to define the LAT characteristics you specify.

```
@SYS$STARTUP:LAT$STARTUP "P1" "P2" "P3" "P4" "P5"
```


LAT New Features

9.1 Starting Up the LAT Protocol Software

Digital recommends that you modify LAT\$SYSTARTUP.COM directly instead of appending these arguments to the @SYS\$STARTUP:LAT\$STARTUP command. However, should you choose to specify these arguments with the @SYS\$STARTUP:LAT\$STARTUP command, note that arguments P1 through P5 are defined as follows:

Format	Meaning
P1 Argument	
Service name	Name of the VMS service. For clustered VMS service nodes, use the cluster alias as the service name. For independent VMS service nodes, use the DECnet node name. SYS\$STARTUP:LAT\$SYSTARTUP.COM uses the argument P1 to assign a service name to the node (with the LATCP CREATE SERVICE command).
P2 Through P4 Arguments¹	
/IDENTIFICATION="string"	Description of the node and its services that is advertised over the Ethernet. The default is the string defined by the logical name SYS\$ANNOUNCE. Make sure you include five sets of quotation marks around the identification string, as in the following example: /IDENTIFICATION="""""Official system center"""".
/GROUPS=(ENABLE=group-list)	Terminal server groups qualified to establish connections with the VMS service node. By default, group 0 is enabled.
/GROUPS=(DISABLE=group-list)	Removes previously enabled terminal server groups. If you are specifying the preceding qualifier to enable groups, you can combine the qualifiers into one, as shown in the example that follows this table.
P5 Argument²	
Any qualifiers valid with the CREATE SERVICE command.	SYS\$STARTUP:LAT\$SYSTARTUP.COM uses this argument to assign service characteristics with the LATCP CREATE SERVICE command. You can specify the /IDENTIFICATION, /LOG, and /STATIC_RATING qualifiers. Specify several qualifiers as shown in the following example: "/IDENTIFICATION="""""Official system node""""/STATIC_RATING=250".

¹Any of these qualifiers can be specified. SYS\$STARTUP:LAT\$SYSTARTUP.COM uses the arguments to assign LAT node characteristics (with the LATCP SET NODE command).

²P5 is used only if P1 is specified.

For example, the following command creates the service OFFICE on the VMS service node MOE:

```
$ @SYS$STARTUP:LAT$STARTUP OFFICE -
_ $ /GROUPS=(ENABLE=(1,4-9),DISABLE=0)
```

In addition, if you want to do any of the following LAT network tasks, you must edit LAT\$SYSTARTUP.COM, as described in Section 9.2:

- Set up LAT printers
- Create special application services

- Set up the node to allow outgoing connections (to support the SET HOST/LAT command)

For more information about the LAT protocol software, see Section 9.3. For a full description of all LATCP commands and qualifiers, see the *VMS LAT Control Program (LATCP) Manual*.

9.2 Site-Specific LAT Command Procedure (LAT\$SYSTARTUP.COM)

The command procedure SYS\$MANAGER:LAT\$SYSTARTUP.COM. contains LATCP commands that define LAT characteristics. LAT\$SYSTARTUP.COM is invoked when you execute the LAT\$STARTUP command procedure. As explained in Section 9.1, you typically execute LAT\$STARTUP.COM from SYSTARTUP_V5.COM.

You do not need to edit LAT\$SYSTARTUP.COM if you want your VMS node to be a LAT service node that only supports incoming connections from interactive terminals. You can assign a service name and other characteristics by specifying parameters when you invoke the command procedure SYS\$STARTUP:LAT\$STARTUP, as described in Section 9.1.

However, you can edit LAT\$SYSTARTUP.COM to add LATCP commands to customize LAT characteristics for your VMS node; for example:

- To create more than one service (see Section 9.2.1)
- To create logical ports for printers (see Section 9.2.2)
- To create logical ports for special application services (see Section 9.2.2)
- To enable outgoing LAT connections to support the SET HOST/LAT command (see Section 9.2.3)
- To tailor VMS node characteristics; for example, to assign special service announcements or Ethernet links (see the *VMS LAT Control Program (LATCP) Manual*)

Note

Do not modify the command procedures LAT\$STARTUP.COM and LAT\$CONFIG.COM. These procedures perform functions necessary for the LAT protocol software to run correctly. Modify only LAT\$SYSTARTUP.COM to customize LAT characteristics for specific sites.

If you edit LAT\$SYSTARTUP.COM, you should add only LATCP commands. In addition, you should conform to the order of commands in the template file SYS\$MANAGER:LAT\$SYSTARTUP.TEMPLATE. The *VMS LAT Control Program (LATCP) Manual* provides a sample edited LAT\$SYSTARTUP procedure and a full description of the commands you can include in LAT\$SYSTARTUP.COM.

9.2 Site-Specific LAT Command Procedure (LAT\$SYSTARTUP.COM)

9.2.1 Creating a VMS Service

The LAT\$SYSTARTUP.COM procedure provided by Digital creates one service. A **service** can be either a primary service, through which users can access all the resources of the computer system, or it can be a special application service, such as a data entry program or an online news service. The procedure creates the service with the same name as that of your VMS node unless you specify a unique service name as an argument to the @SYS\$STARTUP:LAT\$STARTUP.COM command, as explained in Section 9.1.

You can add CREATE SERVICE commands to LAT\$SYSTARTUP.COM to create additional services.

If you create an application service, Digital recommends that you assign the name of the application program. For example, adding the following command to LAT\$SYSTARTUP.COM creates an application service called NEWS on the local node. The /IDENTIFICATION qualifier ensures that this service will be identified in service announcements and in the display generated by the LATCP SHOW NODE command.

```
$ LCP CREATE SERVICE NEWS /IDENTIFICATION /APPLICATION
```

For more information about the LATCP command CREATE SERVICE, see the *VMS LAT Control Program (LATCP) Manual*.

9.2.2 Setting Up Ports

The LAT\$SYSTARTUP.COM procedure provided by Digital includes sample commands to create logical ports on the VMS service node and to associate them with physical ports or services on the terminal server node. These ports can be used for application services and remote printers. Enable these commands by removing the exclamation points (!) that precede them or by adding similar CREATE PORT and SET PORT commands to meet your needs. For information about the LATCP commands CREATE PORT and SET PORT, see the *VMS LAT Control Program (LATCP) Manual*.

Note

Digital strongly recommends that you create application and dedicated ports *after* the LATCP command SET NODE/STATE=ON is executed. This minimizes nonpaged pool memory usage and eliminates the possibility of creating duplicate ports. For more information, see the descriptions of the /DEVICE_SEED and /STATE qualifiers in the SET NODE reference section of the *VMS LAT Control Program (LATCP) Manual*.

Setting Up Printers

If you set up a port for a printer, you must also perform the following tasks:

1. Create a spooled output queue for the printer.
2. Add a command to start the queue to the startup command procedure that starts your queues or to SYSTARTUP_V5.COM.

These tasks are described in the *Guide to Setting Up a VMS System*. For more information about LAT printer queues, see the chapter on batch and print operations in the *Guide to Maintaining a VMS System*.

9.2 Site-Specific LAT Command Procedure (LAT\$SYSTARTUP.COM)

Setting Up Special Application Services

To establish a special application service, include the /DEDICATED qualifier when defining a LAT port. The application program to which the service connects must define the same dedicated port. For example, inserting and then executing the following commands in LAT\$SYSTARTUP.COM sets up ports for an application service called NEWS:

```
$ LCP CREATE PORT LTA333: /DEDICATED
$ LCP SET PORT LTA333: /SERVICE=NEWS
```

Before application services can be available to user terminals on the LAT network, you must start the application program. You usually add commands to do this in SYLOGIN.COM.

9.2.3 Enabling Outgoing LAT Connections

By default, outgoing LAT connections are disabled on a node. If you want to allow users to use the SET HOST/LAT connection to establish LAT connections from the node, you must edit LAT\$SYSTARTUP.COM to enable outgoing connections. For more details on using the SET HOST/LAT command for outgoing LAT connections, see Section 9.3.3.4.

Commands to enable outgoing connections are included in the LAT\$SYSTARTUP.COM procedure provided by Digital. Enable the command of your choice by removing the exclamation point (!) that precedes it or add a similar command to meet your needs. For more information, see the descriptions of the /CONNECTIONS and /USER_GROUPS qualifiers in the SET NODE reference section of the *VMS LAT Control Program (LATCP) Manual*.

To attain optimal SET HOST/LAT performance and forward port performance, set the SYSGEN parameter TTY_ALTYPAMD to 1500 and reboot.

9.3 Connecting to a LAT Network

The VMS operating system uses the LAT communications protocol software to communicate with terminal servers and other systems within a local area network.

Terminal servers are communication devices dedicated for connecting terminals, modems, or printers to an Ethernet network. Terminal servers provide a cost-effective method of connecting many user terminals to a computer. Terminal servers save on cable requirements and they maximize the number of devices that can access a computer.

With the LAT protocol software, the VMS operating system can offer resources (services) that terminal servers can access. A system that offers LAT services is called a service node. In addition, VMS nodes can access LAT services by enabling outgoing connections (using LATCP) and using the SET HOST/LAT command. (In the remainder of this chapter, "servers" refers both to dedicated terminal servers and VMS nodes that allow access to other LAT services.)

9.3.1 Function of the LAT Protocol Software

The LAT protocol is the software that allows terminal server devices and computers to communicate within a local area network (LAN). The LAT protocol software is concerned with matching terminals and other devices to the computing resources (services) of a LAN. Because LAT terminals no longer connect directly to the computer (service node) they are accessing, the local server must listen for service requests from its terminals and be able to match the terminals with computers that provide the desired services.

9.3 Connecting to a LAT Network

Using the LAT protocol software, a VMS operating system announces its available services over the Ethernet. Servers listen to the Ethernet announcements and build a database of service information so that they can locate an appropriate VMS system when a user terminal requests computing services. For example, a user terminal might request general processing service or a data entry program on a VMS operating system. A server uses the LAT protocol software to establish and maintain a connection between the requesting terminal and the VMS operating system.

Sometimes a VMS operating system can request services from a terminal server. The LAT protocol software allows VMS systems to ask for connections to printers or other devices attached to a terminal server.

9.3.2 Advantages of the LAT Protocol Software

The LAT protocol software allows you to make the resources of any computer on a local area network available to any user in that network.

In addition to general processing resources, you can set up terminals, printers, and modems so that they are available from multiple systems in the local area network. This allows you to efficiently use these resources, and to keep them available even if one of the systems in the network must be shut down.

You can also set up application programs, such as data entry programs or news services, as resources. When a user requests a connection to the resource, the LAT protocol software sets up a connection directly to the application program. No login procedure is necessary.

The LAT protocol software provides load balancing features and recovery mechanisms so that users get the best, most consistent service possible. In their broadcast messages, VMS systems rate the availability of their services so that servers can establish connections to computing resources on the least busy node. If a node becomes unavailable for any reason, the servers attempt to provide services on alternate service nodes.

In addition, users can establish multiple computing sessions on their terminals, connecting to several different computers and switching easily from one computing session to another. After switching from one session to another, users can return to the previous session and pick up where they left off. This saves users the time normally required to close out and reopen files or accounts and to return to the same point in a session.

Finally, the LAT protocol software can provide improved system performance. Because the servers bundle messages onto a single Ethernet interface, a server interface decreases the network traffic and reduces the number of computer interrupts encountered in systems where terminals, modems, and printers each have a physical connection to the computer.

9.3.3 The LAT Network

A LAT network is any local area network where terminal servers and operating systems use the LAT protocol software. A LAT network can coexist on the same Ethernet with other protocols. The LAT protocol software, which operates on both terminal servers and the VMS operating system, is designed to ensure the safe transmission of data over the Ethernet.

The LAT network consists of the following entities:

- VMS service nodes
- Terminal server nodes

- VMS nodes allowing outgoing connections
- Ethernet coaxial cable

VMS service nodes supply computing resources for the local network, while terminal server nodes (or VMS nodes allowing outgoing connections) port their terminals, modems, or printers to those resources upon request from a user terminal or an application program.

You can use the LAT Control Program (LATCP) Utility to configure the LAT characteristics for a VMS system. LATCP allows you to set up a VMS system to support:

- Incoming access only
- Outgoing access only
- Both incoming and outgoing access

The VMS systems that support incoming LAT connections are service nodes. (You can also set up a VMS system so that it supports neither incoming nor outgoing access.) See the *VMS LAT Control Program (LATCP) Manual* for more information.

9.3.3.1 VMS Service Nodes

A VMS service node is one type of node in a LAT network. (Nodes that are not using VMS can also be used along with VMS nodes in a LAT network.) A service node is an individual computer in a LAN that offers its resources to users and devices. Because the VMS operating system contains the LAT protocol software, any VMS system can be configured as a service node within a LAT network.

Types of Services

Each VMS node offers its resources as a *service*. Often, a node offers a general processing service, but it can offer special application services as well. Any or all of the services can be specialized applications.

For example, a VMS service node might offer three services: one service for general processing, another for data entry, and a third for stock quotations. The general processing service would allow the use of the general computing environment. The data entry and stock services, on the other hand, would be restricted environments, with connections to the application service but to no other part of the service node.

Each service is distinguished by the name the system manager assigns to it. In a VMS cluster, Digital recommends that the service name be the same as the cluster name. In a standalone system, Digital recommends that the service name be the same as the node name. With special service applications, the service holds the name of the application.

Service Announcements

A VMS service node announces its services over the LAN at regular intervals so that terminal servers (and VMS systems that allow outgoing connections) know about the availability of these network resources. The service announcement provides the physical node name, the service names, a description of services, and a rating of service availability. Servers listen to the Ethernet announcements and record information in a database. On VMS nodes allowing outgoing connections, this database is maintained by the LAT ancillary control process (LATACP).

Whenever a user terminal or application program requests a service, the server node connects to the appropriate VMS service node.

9.3 Connecting to a LAT Network

Print Requests

In some cases, VMS service nodes can request services from terminal servers. The most common situation is when the VMS system wants to use a printer that is ported to a terminal server. VMS submits the print request to the terminal server print queue that is set up and initialized in the VMS startup procedure. Then the LAT symbiont (the process that transfers data to or from mass storage devices) requests the LAT port driver to create and terminate connections to the remote printer.

For information on setting up queues for printers connected to LAT ports, see the chapter on batch and print operations in the *Guide to Maintaining a VMS System*.

9.3.3.2 Terminal Server Nodes

A terminal server node is the second type of node in a LAT network. A terminal server node is usually located near the terminals and printers it supports. The terminals and printers are physically connected to the terminal server; the terminal server is physically connected to the Ethernet.

Locating VMS Service Nodes

Terminal servers build and maintain a directory of services from announcements advertised over the network. Then, when terminal servers receive requests for services from terminal users, they can scan their service database and locate the computer that offers the requested service.

Terminal servers not only look for the VMS node that provides the requested service, they can also evaluate the service rating of that node. If a requested service is offered by more than one node, then the service rating is used to select the node that is least busy. A server establishes a logical connection between the user terminal and the VMS service node.

Setting Up Connections

One logical connection carries all the data directed from one terminal server node to a VMS service node. That is, the server combines data from all terminals communicating with the same VMS node onto one connection. A terminal server establishes a logical connection with a VMS service node only if a logical connection does not already exist.

If a connection fails for any reason, a terminal server attempts to find another node offering the same service and "rolls over" the connection so users can continue their computing sessions.

Even though terminal connections are bundled together, each terminal can be uniquely identified by its name. A terminal name consists of two parts. The first part is the name of the port on the terminal server that the terminal line plugs into. The second part is the name of the terminal server node.

Servicing VMS Nodes

Although terminal servers are usually the requesting nodes in a LAT network, sometimes VMS service nodes request service from terminal servers. Most commonly, a VMS service node queues print requests to remote printers connected to terminal servers.

9.3.3.3 VMS Nodes Allowing Incoming and Outgoing Connections

VMS nodes can be set up to allow incoming connections, outgoing connections, or both. These VMS nodes locate service nodes and set up connections as do terminal server nodes. The database of information about available nodes and services is maintained by the LAT ancillary control process (LATACP).

On a VMS node that is set up to allow outgoing LAT connections, a user can connect to another node in the LAT network by entering the SET HOST/LAT command. The following section describes how to use this new command.

9.3.3.4 Using the SET HOST/LAT Command

The SET HOST/LAT command allows you to connect your terminal to a specified service, establishing one LAT session for communication between your terminal and that service.

The service node that provides the service must be on a remote node, must be on the same extended LAN, and must be running at least Version 5.0 of the LAT protocol software. (Note that you cannot use SET HOST/LAT to connect to the local node.)

Some services are protected with passwords. You are prompted for a password unless you specify the password with the /PASSWORD qualifier.

Once the connection to the service is made, you can interact with the service as if your terminal were connected directly to it. Some services will prompt you. For example, if the service is a VMS system, it prompts you for a user name and password. You must have an account on the service node in order to log in.

Press the disconnect character to end the LAT session and return to DCL command level on your local system. With some services, such as general timesharing services like VMS, you can end the LAT session by logging out of the service. The default disconnect character is Ctrl/\ . Use the /DISCONNECT qualifier to change the default disconnect character.

The format for entering this command is as follows:

SET HOST/LAT service-name

Note that *service-name* specifies the name of the service to which you want your terminal connected. If several service nodes offer the same service and you do not specify the /NODE=node-name qualifier, your terminal connects to the service node that is least busy.

To display a list of services on your LAN, use the LAT Control Program (LATCP) SHOW SERVICES command, as described in the *VMS LAT Control Program (LATCP) Manual*.

The qualifiers you can specify for the SET HOST/LAT command are as follows:

- /[NO]AUTOCONNECT

Specifies whether connection attempts should be retried automatically when a connection fails because a service is unknown or unavailable or because a node is unknown or unreachable. Also specifies that reconnecting should be attempted automatically if a service has disconnected abnormally. The default is /NOAUTOCONNECT.

9.3 Connecting to a LAT Network

- **/BREAK=break-character**
Defines a character that generates a break on lines that expect a break rather than a carriage return. To generate a break, press `Ctrl/break-character`. You can select any ASCII character from @ through Z, except C, M, Q, S, Y, and the left bracket ([). You cannot select a character that is already defined as the disconnect character.
- **/DESTINATION_PORT=port-name**
Specifies the port on a node to which you want to connect. The **/NODE** qualifier is required when you specify the **/DESTINATION_PORT** qualifier. The port must be available and must offer the service you specify. VMS and certain other LAT service node systems ignore the **/DESTINATION_PORT** qualifier.
- **/DISCONNECT=disconnect-character**
Defines the character that you can use to disconnect from a remote session. The default disconnect character is `Ctrl/\`. To generate a disconnect, press `Ctrl/disconnect-character`. You can select any ASCII character from @ through Z, except C, M, Q, S, Y, and the left bracket ([). For example, if you specify **/DISCONNECT=A**, `Ctrl/A` will be the disconnect character. You cannot select a character that is already defined as the break character.
- **/LOG[=log-file]**
Logs all data that is delivered during the LAT session. If you do not specify a name for the log file, the data is stored in the file `SETHOST_LAT.LOG`.
- **/NODE=node-name**
Specifies the node that offers the service to which you want to connect. The node you specify must be a remote node. Failover is not performed if the connection fails.
- **/PASSWORD=password**
Specifies the password required by a service that is protected with a password. If you do not specify the **/PASSWORD** qualifier when requesting a connection to such a service, you are prompted for the password.

The following examples illustrate how to use the `SET HOST/LAT` command:

Examples

```
1. $ SET HOST/LAT SORTER
%LAT-S-CONNECTED, session to SORTER established
%LAT-I-TODISCON, type ^\ to disconnect the session
Username: SMITH
Password:
.
.
$ LOGOUT
SMITH logged out at 9-JUL-1991 11:04:51.45
%LAT-I-DISCONNECTED, session disconnected from SORTER
-LAT-I-END, control returned to node HOME
$
```

This `SET HOST/LAT` command connects the user to the service `SORTER`, which is a computer system. The first message confirms that the user has been connected to that service. The second message informs the user how to disconnect the session. (The user can also disconnect the session by logging

out from SORTER.) SORTER then prompts for the user name and password. Use the normal login procedure to log in to the computer system. When the user logs out of the service SORTER, the terminal displays the DCL command prompt of the user's local system (HOME).

2. \$ SET HOST/LAT/DESTINATION_PORT=BOSTON/NODE=STATE/DISCONNECT=F BUDGET

This command connects the user's terminal to the service BUDGET that is offered on port BOSTON on service node STATE. The user can disconnect the session by typing Ctrl/F.

3. \$ SET HOST/LAT PURSE
Password:

This command attempts to connect the user's terminal to the service PURSE. The service PURSE is protected, so the user is prompted for a password. The user could have specified the password within the SET HOST/LAT command, as shown in the next example.

4. \$ SET HOST/LAT/PASSWORD=BEOR PURSE

This command connects the user's terminal to the service PURSE. The password is BEOR.

9.3.3.5 A Sample LAT Configuration

Figure 9-1 illustrates the components of a LAT network. The network consists of an Ethernet cable connecting VMS service nodes and terminal server nodes.

The three VMS service nodes in Figure 9-1, named MOE, LARRY, and ALEXIS, each offer services to terminal server nodes on the network.

Two of the VMS service nodes, MOE and LARRY, belong to the OFFICE cluster. (The cluster is distinguished by its computer interconnect (CI) and star coupler.) Because MOE and LARRY are clustered, their service names are the same as their cluster name. Because both VMS service nodes offer an OFFICE service, terminal server nodes can assess the work load on both OFFICE nodes and establish a connection to a node that offers the service that is least busy.

The third VMS service node, ALEXIS, is an independent node in the LAT network, so its service name is the same as its node name.

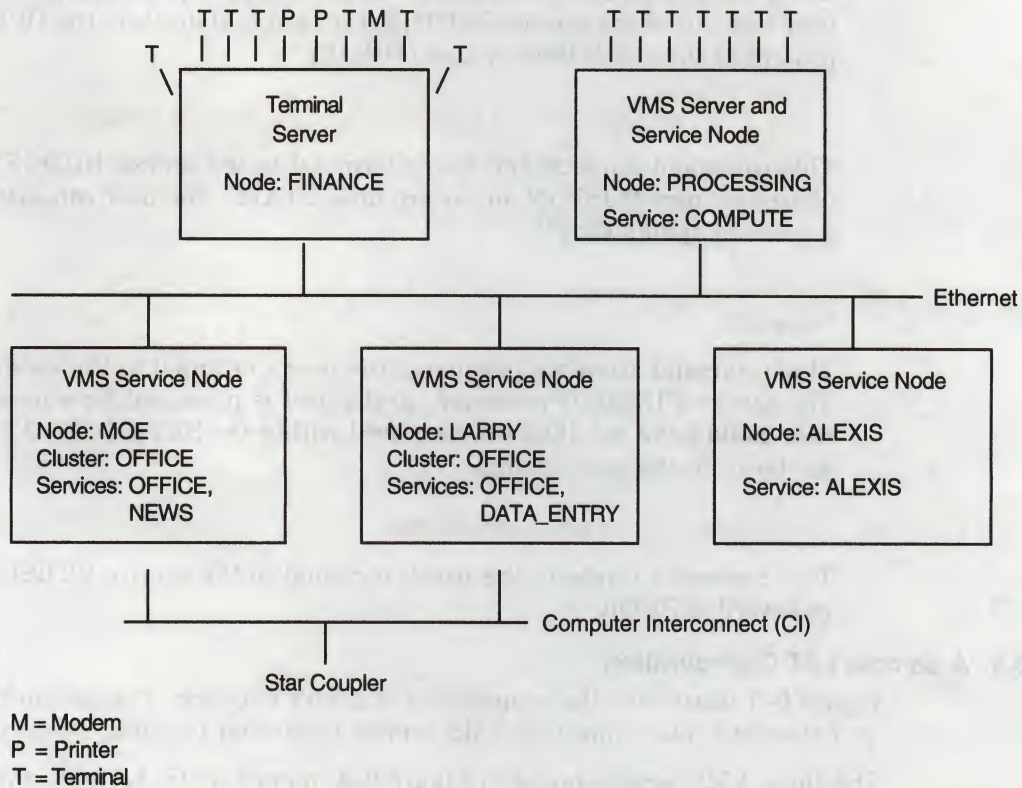
In addition to its primary OFFICE service, node MOE offers an application service called NEWS. With this specialized service, user terminals can connect directly to the online news program, without any login procedure but also without general access to the general computer resources of the node.

The node FINANCE, shown in Figure 9-1, is a terminal server node. The node PROCESSING is a VMS node allowing outgoing connections. Node FINANCE supports a number of interactive terminals as well as a modem and a printer. This node can accept print requests from any of the three VMS service nodes, provided each of the service nodes has set up print queues to support remote printers on the terminal server.

Node PROCESSING is a VMS server and service node that offers the COMPUTE service.

9.3 Connecting to a LAT Network

Figure 9-1 A LAT Network Configuration



ZK-1110A-GE

9.3.3.6 LAT Relationship to VMS Clusters and DECnet

Although the LAT protocol software works independently of VMS VAXcluster software, Digital recommends that you configure a VMS service node to complement the cluster concept. You achieve this by creating a service on each node in a cluster and assigning the cluster name to this service. A terminal server assesses the availability of cluster services and establishes a connection to the node that is least busy. Thus, the LAT protocol software helps balance the cluster load. If one node in the cluster fails, the terminal server can transfer the failed connections to another service node within the cluster.

LAT does not use DECnet as a message transport facility, but instead uses its own virtual circuit layer to implement a transport mechanism. Essentially, LAT and DECnet work independently in a common Ethernet environment. For compatibility, if a VMS service node is also a DECnet node, the VMS service node name should be the same as the DECnet node name.

9.3.4 Summary of LAT System Management Tasks

The following sections summarize tasks you perform to manage the new LAT protocol software. Before performing these tasks, however, review the *VMS Version 5.5 Release Notes* for additional LAT information.

9.3.4.1 Starting Up the LAT Protocol Software

As system manager, you start up the LAT protocol software and configure your node as a VMS service node by executing the command procedure `SY$STARTUP:LAT$STARTUP`. This procedure executes the following procedures:

- `LAT$CONFIG.COM`, to load the LAT terminal driver `LTDRIVER` and create the `LATACP` process
- `LAT$SYSTARTUP.COM`, to execute `LATCP` commands that define LAT characteristics

To make sure the LAT protocol software is started each time the system boots, add a command to execute this procedure in the site-specific command procedure `SYSTARTUP_V5.COM`. For instructions, see Section 9.1.

9.3.4.2 Customizing LAT Characteristics

To define special LAT characteristics for your node, edit the site-specific command procedure `SY$MANAGER:LAT$SYSTARTUP.COM`, as described in Section 9.2.

If you only want to set up your node as a service node with incoming connections enabled, you do not need to edit `LAT$SYSTARTUP.COM`. However, you might edit `LAT$SYSTARTUP.COM` to do one or more of the following tasks:

- Create more than one service on a node
- Create special application services
- Set up LAT printers
- Enable outgoing LAT connections (to allow a VMS node to act as a terminal server node)
- Tailor VMS node characteristics; for example, to assign special service announcements or LAN links (connections to Ethernet or FDDI¹ devices, for example)

Caution

Do not edit the `LAT$STARTUP.COM` or `LAT$CONFIG.COM` procedures.

9.3.4.3 Using LATCP to Control the LAT Protocol Software

The LAT Control Program Utility (`LATCP`) serves as a command interface to the LAT software running on the VMS node. `LATCP` commands allow you to stop and start the LAT driver (`LTDRIVER`) and to modify and display LAT characteristics of the VMS node.

For detailed information about all `LATCP` commands and qualifiers, see the *VMS LAT Control Program (LATCP) Manual*. See the *VMS Version 5.5 Release Notes* for information about `LATCP` commands and qualifiers that are now obsolete.

¹ Fiber distributed data interface

9.3.4.4 Managing the LATACP Database Size

On VMS nodes, the LATACP process maintains a database of available nodes and services. The nodes and services may be those that are multicast (announced on the LAN) from remote LAT nodes, or they could consist of the local node and one or more local services that you create on your own system. The maximum size of this database is dependent on the SYSGEN parameter CTLPAGES.

After you enter a LATCP command, you might get the following response:

```
%LAT-W-CMDERROR, error reported by command executor
-LAT-F-ACPNOCTL, insufficient resources - ACP CTL/P1 space limit reached
```

If so, this signifies that the database size has reached the CTLPAGES limit. You can correct the situation in one of the following ways:

- Reduce the size of the database by reducing the node limit. Use the LATCP command SHOW NODE to display the node limit; use the command SET NODE/NODE_LIMIT to change it. For more information, see the *VMS LAT Control Program (LATCP) Manual*.
- Reduce the size of the database by reducing the user group codes that are enabled on the node. Use the LATCP command SHOW NODE to display the enabled user group codes; use the command SET NODE/USER_GROUPS=DISABLE to disable some of them. For more information, see the *VMS LAT Control Program (LATCP) Manual*.

If you choose this option, you must also edit your startup procedures to change the user groups that are enabled each time the system reboots. For more information, see Section 9.2.

- Extend the size of the database by increasing the size of CTLPAGES using SYSGEN (and then rebooting the system). As a general rule, note that every unit of CTLPAGES that you increase is roughly equivalent to six additional nodes or services that will be stored in the database.

VMS License Management Facility

The VMS License Management Facility (LMF) has been enhanced to include a number of new features. This section briefly describes most of these enhancements, but the reader should consult the *VMS License Management Utility Manual* for complete, detailed information.

10.1 Moving and Copying Licenses

The new LICENSE COPY and LICENSE MOVE commands allow the transfer of licenses between databases.

The LICENSE MOVE command creates a new license registration in the target license database and then deletes the license record and its history records from the source database.

The LICENSE COPY command creates a new license registration in the target license database, disables the license record in the source database, and retains the history records in the source database.

Note that the LICENSE MOVE and LICENSE COPY commands do not transfer to the target database any user-supplied data such as reservation lists, modified termination dates, modified units, include or exclude node lists, or comments.

10.2 Deleting Licenses

The new LICENSE DELETE command allows you to delete a license and its history records from a license database.

10.3 Automating License Registration

The LICENSE ISSUE command now accepts the /PROCEDURE qualifier. This qualifier causes the LICENSE ISSUE command to produce output formatted such that it can be invoked as a DCL command procedure to register a license in another license database.

10.4 Creating License Reservation Lists

The LICENSE MODIFY command now accepts the /RESERVE qualifier, which allows system managers or privileged users to attach a list of names to a product license. This list of names, called a reservation list, restricts use of the product to the names in the list.

You can add a reservation list to *any* Product Authorization Key (PAK).

The following example shows how to add a reservation list to a product license using the MODIFY/RESERVE command:

```
$ LICENSE MODIFY FORTRAN/RESERVE=(DOE,SMITH,JONES)
$ LICENSE UNLOAD FORTRAN
$ LICENSE LOAD FORTRAN
```


VMS License Management Facility

10.4 Creating License Reservation Lists

This command example restricts the use of VAX FORTRAN to the users named Doe, Smith, and Jones.

10.5 Support for PAKs with the RESERVE_UNITS Option

LMF now allows software vendors to issue PAKs with the RESERVE_UNITS option. This option may be used by license issuers whose terms and conditions require that use of the product be restricted to a specified number of named users.

When registering a PAK that makes use of the RESERVE_UNITS option, a customer must specify a reservation list. The number of names in this list must be no larger than that allowed by the product license. Use the LICENSE MODIFY/RESERVE command to associate this reservation list with the product license.

Note

At this time, PAKs making use of this option can be registered and used only on systems that are running VMS Version 5.5 or are running VMS Versions 5.2 to 5.4-3, inclusive, but have separately installed LMF Version 1.1.

10.6 Ease-of-Use Features

To ease license management, license managers can now perform operations on groups of licenses. For example, to disable five different FORTRAN licenses on a single machine, you can now issue a single command instead of five separate commands.

The ability to operate on groups of licenses is provided by support for the following:

- Standard VMS wildcard characters (* and %), which may be used with most parameters and qualifiers (see the *VMS License Management Utility Manual* for details)
- Lists of product names in the *product-name* parameter of most commands
- A new /ALL qualifier that expands the command operation to affect all licenses that match the specification provided

10.7 Revised SYS\$UPDATE:VMSLICENSE.COM

The command procedure SYS\$UPDATE:VMSLICENSE.COM has been significantly expanded to include support for most of the new features available. Information about the new features is included at the beginning of the command procedure and is available to you when you issue the following command:

```
$ @SYS$UPDATE:VMSLICENSE
```

For additional information about support for the new features, see the *VMS License Management Utility Manual*.

Movefile Command Qualifiers

This chapter describes the SET FILE command qualifiers and the enhancements to the DCL commands DIRECTORY/FULL, DUMP/HEADER, and DUMP/FILE_HEADER that support movefile operations. It also lists the system files for which movefile operations are automatically disabled.

Programming support for movefile operations is presented in Chapter 22.

11.1 SET FILE Command Qualifiers

This section details the SET FILE command qualifiers that support movefile operations. The information is presented in the format used for documenting DCL commands.

Example

11.3 Critical System Files

This section lists the system files for which movefile operations are automatically disabled. The list applies to all system disks.

- [000000]QUORUM.DAT
- [SYS*...SYS\$LDR]CPULOA.EXE
- [SYS*...SYS\$LDR]DDIF\$RMS_EXTENSION.EXE
- [SYS*...SYS\$LDR]ERRORLOG.EXE
- [SYS*...SYS\$LDR]EVENT_FLAGS_AND_ASTS.EXE
- [SYS*...SYS\$LDR]EXCEPTION.EXE
- [SYS*...SYS\$LDR]EXEC_INIT.EXE
- [SYS*...SYS\$LDR]FPEMUL.EXE
- [SYS*...SYS\$LDR]IMAGE_MANAGEMENT.EXE
- [SYS*...SYS\$LDR]IO_ROUTINES.EXE
- [SYS*...SYS\$LDR]LMF\$GROUP_TABLE.EXE
- [SYS*...SYS\$LDR]LOCKING.EXE
- [SYS*...SYS\$LDR]LOGICAL_NAMES.EXE
- [SYS*...SYS\$LDR]MESSAGE_ROUTINES.EXE
- [SYS*...SYS\$LDR]PAGE_MANAGEMENT.EXE
- [SYS*...SYS\$LDR]PRIMITIVE_IO.EXE
- [SYS*...SYS\$LDR]PROCESS_MANAGEMENT.EXE
- [SYS*...SYS\$LDR]RECOVERY_UNIT_SERVICES.EXE
- [SYS*...SYS\$LDR]RMS.EXE
- [SYS*...SYS\$LDR]SECURITY.EXE
- [SYS*...SYS\$LDR]SYS.EXE
- [SYS*...SYS\$LDR]SYS\$CLUSTER.EXE
- [SYS*...SYS\$LDR]SYS\$NAME_SERVICES.EXE
- [SYS*...SYS\$LDR]SYS\$NETWORK_SERVICES.EXE
- [SYS*...SYS\$LDR]SYS\$SCS.EXE
- [SYS*...SYS\$LDR]SYS\$TRANSACTION_SERVICES.EXE
- [SYS*...SYS\$LDR]SYSDEVICE.EXE
- [SYS*...SYS\$LDR]SYSGETSYL.EXE
- [SYS*...SYS\$LDR]SYSLDR_DYN.EXE
- [SYS*...SYS\$LDR]SYSLICENSE.EXE
- [SYS*...SYS\$LDR]SYSLOA*.EXE
- [SYS*...SYS\$LDR]SYSTEM_DEBUG.EXE
- [SYS*...SYS\$LDR]SYSTEM_PRIMITIVES.EXE
- [SYS*...SYS\$LDR]SYSTEM_SYNCHRONIZATION.EXE

- [SYS*...SYS\$LDR]SYSTEM_SYNCHRONIZATION_MIN.EXE
- [SYS*...SYS\$LDR]SYSTEM_SYNCHRONIZATION_UNI.EXE
- [SYS*...SYS\$LDR]VAXEMUL.EXE
- [SYS*...SYS\$LDR]VECTOR_PROCESSING.EXE
- [SYS*...SYS\$LDR]VMS\$SYSTEM_IMAGES.DATA
- [SYS*...SYS\$LDR]WORKING_SET_MANAGEMENT.EXE
- [SYS*...SYS\$LDR]CWDRIVER.EXE
- [SYS*...SYS\$LDR]DBDRIVER.EXE
- [SYS*...SYS\$LDR]DDDRIVER.EXE
- [SYS*...SYS\$LDR]DLDRIVER.EXE
- [SYS*...SYS\$LDR]DMDRIVER.EXE
- [SYS*...SYS\$LDR]DRDRIVER.EXE
- [SYS*...SYS\$LDR]DSDRIVER.EXE
- [SYS*...SYS\$LDR]DUDRIVER.EXE
- [SYS*...SYS\$LDR]DXDRIVER.EXE
- [SYS*...SYS\$LDR]EFDRIVER.EXE
- [SYS*...SYS\$LDR]EPDRIVER.EXE
- [SYS*...SYS\$LDR]GDDRIVER.EXE
- [SYS*...SYS\$LDR]P*DRIVER.EXE
- [SYS*...SYS\$LDR]SHDRIVER.EXE
- [SYS*...SYS\$LDR]TTDRIVER.EXE
- [SYS*...SYS\$LDR]UNKDRIVER.EXE
- [SYS*...SYS\$LDR]X*DRIVER.EXE
- [SYS*...SYSEXE]CLUSTER_AUTHORIZE.DAT
- [SYS*...SYSEXE]F11BXQP.EXE
- [SYS*...SYSEXE]LOGINOUT.EXE
- [SYS*...SYSEXE]PAGEFILE.SYS
- [SYS*...SYSEXE]STABACKUP.EXE
- [SYS*...SYSEXE]STACONFIG.EXE
- [SYS*...SYSEXE]SWAPFILE.SYS
- [SYS*...SYSEXE]SYS\$INCARNATION.DAT
- [SYS*...SYSEXE]SYSBOOT.EXE
- [SYS*...SYSEXE]SYSBOOT_XDELTA.EXE
- [SYS*...SYSEXE]SYSDUMP.DMP
- [SYS*...SYSEXE]*.PAR
- [SYS*...SYSEXE]VMB.EXE

11.3 Critical System Files

- [SYS*...SYSMAINT]DIAGBOOT.EXE

- [SYS*...SYSMSG]SYSMSG.EXE

Part IV

Programming Features

This part provides information about new programming features introduced with VMS Version 5.5. The following chapters contain the programming information:

- Chapter 12, System Service Support for the VMS Batch and Print Queuing System
- Chapter 13, Run-Time Library Routines
- Chapter 14, VMS Debugger: Tasking and Multithread Support
- Chapter 15, DECthreads
- Chapter 16, DECdtm System Services: New and Changed Features
- Chapter 17, LAT \$QIO Functions
- Chapter 18, Asynchronous Printer Support
- Chapter 19, Support for Case Sensitivity
- Chapter 20, System Dump Analyzer
- Chapter 21, Mailbox Driver
- Chapter 22, \$QIO Support for Moving Disk Files

System Service Support for the VMS Batch and Print Queuing System

This chapter provides a summary of system service changes that support the VMS batch and print queuing system introduced in VMS Version 5.5.

12.1 \$GETQUI and \$SNDJBC System Services

The system services \$GETQUI and \$SNDJBC have been enhanced to support new features introduced with the new batch and print queuing system. For detailed information about new system service features, see the *VMS System Services Reference Manual*. The following sections list the system service changes that support the batch and print queuing system. Where applicable, the listed system service changes include a parenthetical reference to related information in other sections of this manual.

12.1.1 \$GETQUI Service

Changes to the \$GETQUI system service include the following:

Five new item codes:

- QUI\$_AUTOSTART_ON (autostart feature — Section 5.4)
- QUI\$_JOB_RETENTION_TIME (user-specified job retention — Section 3.3)
- QUI\$_JOB_COMPLETION_TIME (change to SHOW ENTRY display — Section 3.1.1)
- QUI\$_JOB_COMPLETION_QUEUE (change to SHOW ENTRY display— Section 3.1.1)
- QUI\$_SEARCH_JOB_NAME (new job name parameter for SHOW ENTRY command — Section 3.1.2)

Eight new bit codes:

- QUI\$V_JOB_RETENTION (user-specified job retention — Section 3.3)
- QUI\$V_JOB_ERROR_RETENTION (user-specified job retention — Section 3.3)
- QUI\$V_QUEUE_AVAILABLE
- QUI\$V_QUEUE_BUSY
- QUI\$V_QUEUE_STOP_PENDING
- QUI\$V_JOB_STALLED (new stalled job state — Section 3.1.3)
- QUI\$V_QUEUE_AUTOSTART (autostart feature — Section 5.4)
- QUI\$V_QUEUE_AUTOSTART_INACTIVE (autostart feature — Section 5.4)

Designation of nine previously existing QUI\$_QUEUE_STATUS bits as state bits:

- QUI\$V_QUEUE_IDLE
- QUI\$V_QUEUE_DISABLED
- QUI\$V_QUEUE_PAUSED
- QUI\$V_QUEUE_PAUSING
- QUI\$V_QUEUE_RESUMING

System Service Support for the VMS Batch and Print Queuing System

12.1 \$GETQUI and \$SNDJBC System Services

QUI\$V_QUEUE_STALLED
QUI\$V_QUEUE_STARTING
QUI\$V_QUEUE_STOPPED
QUI\$V_QUEUE_STOPPING

12.1.2 \$SNDJBC Service

Changes to the \$SNDJBC system service include the following:

Three new function codes:

SJC\$_STOP_ALL_QUEUES_ON_NODE (new queue manager — Section 5.3.3)

SJC\$_ENABLE_AUTOSTART (autostart feature — Section 5.4)

SJC\$_DISABLE_AUTOSTART (autostart feature — Section 5.4)

Seven new item codes:

SJC\$_QUEUE_MANAGER_NODES (new queue manager — Section 5.3)

SJC\$_QUEUE_DIRECTORY (new queue database — Section 5.2)

SJC\$_AUTOSTART_ON (autostart feature — Section 5.4)

SJC\$_JOB_RETAIN (user-specified job retention — Section 3.3)

SJC\$_JOB_ERROR_RETAIN (user-specified job retention — Section 3.3)

SJC\$_JOB_DEFAULT_RETAIN (user-specified job retention — Section 3.3)

SJC\$_JOB_RETAIN_TIME (user-specified job retention — Section 3.3)

Extended use of one item code:

SJC\$_SCSNODE_NAME

Run-Time Library Routines

This chapter describes new features of the Run-Time Library (RTL). Section 13.1 discusses enhancements to the LIB\$GETQUI library routine. Section 13.2 discusses enhancements to the MTH\$ library and Section 13.3 discusses enhancements to the PPL\$ library.

13.1 LIB\$GETQUI Run-Time Library Routine

New features for the \$GETQUI system service affect the LIB\$GETQUI run-time library routine. For information about the \$GETQUI new features, see the section on \$GETQUI in the *VMS System Services Reference Manual*.

13.2 Fast-Vector Math Routines

This section describes the **fast-vector** math routines, which offer significantly higher performance at the cost of slightly reduced accuracy when compared with corresponding standard-vector math routines. Note too, that some fast-vector math routines have restricted argument domains.

When you specify the compile command qualifiers /VECTOR and /MATH_LIBRARY=FAST, VAX FORTRAN-HPO Version 1.2 selects the appropriate fast-vector math routine, if one exists. The default is /MATH_LIBRARY=ACCURATE. You must specify the /G_FLOATING compile qualifier with the /MATH_LIBRARY=FAST and /VECTOR qualifiers to access the G_floating versions from VAX FORTRAN-HPO. See the *VAX FORTRAN-HPO Version 1.2 Release Notes* for more information.

You can call these routines from VAX MACRO using the standard calling method. The math function names, together with corresponding entry points of the fast-vector math routines, are listed in Table 13-1.

Table 13-1 Fast-Vector Math Routines

Function Name	Data Type	Entry Point
ATAN	F_floating	MTH\$VYATAN_R0_V3
ATAN	D_floating	MTH\$VYDATAN_R0_V5
ATAN	G_floating	MTH\$VYGATAN_R0_V5
ATAN2	F_floating	MTH\$VVYATAN2_R0_V5
ATAN2	D_floating	MTH\$VVYDATAN2_R0_V5
ATAN2	G_floating	MTH\$VVYGATAN2_R0_V5
COS	F_floating	MTH\$VYCOS_R0_V3

(continued on next page)

Run-Time Library Routines

13.2 Fast-Vector Math Routines

Table 13–1 (Cont.) Fast-Vector Math Routines

Function Name	Data Type	Entry Point
COS	D_floating	MTH\$VYDCOS_R0_V3
COS	G_floating	MTH\$VYGCOS_R0_V3
EXP	F_floating	MTH\$VYEXP_R0_V4
EXP	D_floating	MTH\$VYDEXP_R0_V6
EXP	G_floating	MTH\$VYGEXP_R0_V6
LOG	F_floating	MTH\$VYALOG_R0_V5
LOG	D_floating	MTH\$VYDLOG_R0_V5
LOG	G_floating	MTH\$VYGLOG_R0_V5
LOG10	F_floating	MTH\$VYALOG10_R0_V5
LOG10	D_floating	MTH\$VYDLOG10_R0_V5
LOG10	G_floating	MTH\$VYGLOG10_R0_V5
SIN	F_floating	MTH\$VYSIN_R0_V3
SIN	D_floating	MTH\$VYDSIN_R0_V3
SIN	G_floating	MTH\$VYGSIN_R0_V3
SQRT	F_floating	MTH\$VYSQRT_R0_V4
SQRT	D_floating	MTH\$VYDSQRT_R0_V4
SQRT	G_floating	MTH\$VYGSQRT_R0_V4
TAN	F_floating	MTH\$VYTAN_R0_V3
TAN	D_floating	MTH\$VYDTAN_R0_V3
TAN	G_floating	MTH\$VYGTAN_R0_V3
Power (X**Y)	F_floating	OTS\$VYPOWRR_R1_V4
Power (X**Y)	D_floating	OTS\$VYPOWDD_R1_V8
Power (X**Y)	G_floating	OTS\$VYPOWGG_R1_V9

13.2.1 Exception Handling

The fast-vector math routines signal all errors except “floating underflow.” No intermediate calculations result in exceptions. To optimize performance, the following message signals all errors:

%SYSTEM-F-VARITH, vector arithmetic fault

13.2.2 Special Restrictions on Input Arguments

The special restrictions listed in Table 13–2 apply only to the fast-vector routines SIN, COS, and TAN. The standard-vector routines handle the full range of VAX floating-point numbers.

Table 13–2 Input Argument Restrictions

Function Name	Input Argument Domain (in radians)
SIN	~ -6746518783.0, 6746518783.0

(continued on next page)

Table 13–2 (Cont.) Input Argument Restrictions

Function Name	Input Argument Domain (in radians)
COS	~ -6746518783.0, 6746518783.0
TAN	~ -3373259391.5, 3373259391.5

If the application program uses arguments outside of the listed domain, the routine returns the following error message:

%SYSTEM-F-VARITH, vector arithmetic fault

If the application requires argument values beyond the listed limits, use the corresponding standard-vector math routine.

13.2.3 Accuracy

The fast-vector math routines do *not* guarantee the same results as those obtained with the corresponding standard-vector math routines. Calls to the fast-vector routines generally yield results that are different from the scalar and the original vector MTH\$ library routines. The typical maximum error is a 2-LSB (least significant bit) error for the F_floating routines, and a 4-LSB error for the D_floating and G_floating routines. This generally corresponds to a difference in the sixth significant decimal digit for the F_floating routines, the fifteenth digit for D_floating, and the fourteenth digit for G_floating.

13.2.4 Performance

The fast-vector math routines generally provide performance improvements over the standard-vector routines ranging from 15% to 300%, depending on the routines called and input arguments to the routines. The overall performance improvement using fast-vector math routines in a typical user application will increase but not at the same level as the routines themselves. You should do performance and correctness testing of your application using both the fast-vector and the standard-vector math routines before deciding on which to use for your application.

13.3 Parallel Processing Routines

This section describes new features of the Parallel Processing (PPL\$) run-time library.

Changes to the PPL\$ library consist of enhanced unique naming functionality and spin/wait options for several of the blocking synchronization routines.

13.3.1 Enhancements for Unique Naming

PPL\$UNIQUE_NAME now allows a greater degree of unique naming within and among PPL\$ applications.

The default action for PPL\$UNIQUE_NAME has been to take a string and return a new string unique to the application. By calling PPL\$UNIQUE_NAME with the same input string from any process in an application, the user can get the same “application-unique” name returned. Calling the function from another application, with the same input string, results in a string that differs from the string returned to the previous application.

It is also now possible to request that PPL\$UNIQUE_NAME return a string unique to a process. By specifying the new PPL\$M_PROC_UNIQUE flag, the user will receive a "process-unique" name. That is, each time the user supplies the same string to PPL\$UNIQUE_NAME within a process, the same unique string will be returned. If the user specifies the same input string in another process, a different string will be returned, one which is unique to the other process.

In addition to "process-unique" names, the user may now request that a name be made "call-unique." When you specify the PPL\$M_CALL_UNIQUE flag, PPL\$UNIQUE_NAME produces a different return string each time it is called, regardless of the process or the application from which it is called.

13.3.2 Spin/Wait Options for Blocking Synchronization

PPL\$WAIT_AT_BARRIER, PPL\$DECREMENT_SEMAPHORE, and PPL\$REMOVE_WORK_ITEM all now have spin/wait options. A user may request to have a process spin instead of hibernating while it is blocked on the synchronization object. In addition, the user may specify the maximum number of spins to be performed before hibernating.

Two new flags have been added to the PPL\$ library for these options:

- PPL\$M_SPIN_WAIT
Causes the process to spin as long as it is blocked on the synchronization object (never hibernate).
- PPL\$M_SPIN_COUNTED
Causes the process to spin the specified number of times and then hibernate.

VMS Debugger: Tasking and Multithread Support

For VMS Version 5.5, the VMS Debugger provides enhanced support for tasking programs. Tasking programs (also called multithread programs) have multiple threads of execution within a VMS process.

Ada programs have built-in tasking services, and debugger support for VAX Ada tasking programs has been available since VMS Version 4.2 (since VAX Ada Version 1.0).

Starting with VMS Version 5.5, debugger tasking support has been extended to include any program that uses DECthreads or POSIX 1003.4a services. These services are provided for languages that do not have built-in tasking services.

Debugger tasking support enables you to perform functions such as the following:

- Display task information
- Modify task characteristics to control task execution, priority, state transitions, and so on
- Monitor task-specific events and state transitions

14.1 Command Interface: Enhanced Commands and Qualifiers

There are no new commands or qualifiers. However, the following commands, which are task related, have been enhanced to provide the new support:

- SET TASK, SHOW TASK
- SET EVENT_FACILITY (you can now specify THREADS, in addition to ADA and SCAN, as a command parameter)
- SHOW EVENT_FACILITY
- SET BREAK/EVENT, SET TRACE/EVENT (THREADS events are now defined in addition to ADA and SCAN events)

See the *VMS Debugger Manual* for complete information about these commands and qualifiers.

14.2 DECwindows Interface: Enhancements

There are no visible changes to the debugger's DECwindows interface. However, the tasking features that are available by choosing Tasks... from the Data menu in the main window have been enhanced to provide the new support.

See the *VMS Debugger Manual* and online help that is available from the debugger's DECwindows interface for complete information about these features.

VMS Debugger: Tasking and Multithread Support

The VMS Debugger supports tasking and multithreaded execution. This section describes the debugger's support for these features.

The debugger supports tasking and multithreaded execution on VMS systems running on Alpha and VAX architectures. The debugger supports the following features:

• Tasking: The debugger supports tasking on VMS systems running on Alpha and VAX architectures. The debugger supports the following features:

• Multithreading: The debugger supports multithreading on VMS systems running on Alpha and VAX architectures. The debugger supports the following features:

• Tasking and Multithreading: The debugger supports tasking and multithreading on VMS systems running on Alpha and VAX architectures. The debugger supports the following features:

1.1 Command Interface: Enhanced Commands and Qualifiers

The VMS Debugger provides enhanced commands and qualifiers for tasking and multithreading. The following table lists the enhanced commands and qualifiers:

• **Tasking Commands:** The following table lists the enhanced commands for tasking:

• **Multithreading Commands:** The following table lists the enhanced commands for multithreading:

• **Tasking and Multithreading Commands:** The following table lists the enhanced commands for tasking and multithreading:

• **Tasking and Multithreading Qualifiers:** The following table lists the enhanced qualifiers for tasking and multithreading:

• **Tasking and Multithreading Qualifiers:** The following table lists the enhanced qualifiers for tasking and multithreading:

• **Tasking and Multithreading Qualifiers:** The following table lists the enhanced qualifiers for tasking and multithreading:

• **Tasking and Multithreading Qualifiers:** The following table lists the enhanced qualifiers for tasking and multithreading:

1.2 Debugger Interface: Enhanced Commands

The VMS Debugger provides enhanced commands for tasking and multithreading. The following table lists the enhanced commands:

• **Tasking Commands:** The following table lists the enhanced commands for tasking:

• **Multithreading Commands:** The following table lists the enhanced commands for multithreading:

• **Tasking and Multithreading Commands:** The following table lists the enhanced commands for tasking and multithreading:

DECthreads, Digital's multithreading run-time library, contains portable routines used for creating and controlling multiple threads of execution within the address space provided by a single process.

DECthreads is documented in the *Guide to DECthreads*.

15.1 Overview

Threads are used to improve the performance (throughput, computational speed, responsiveness—or some combination) of a program. Multiple threads improve program performance on single-processor systems by permitting the overlap of input and output or other slow operations with computational operations.

Threads are especially advantageous in a network client/server environment. A server receives requests, processes them (often involving a waiting step, for example waiting for a disk read), and sends replies. By creating a thread for each request, the server can improve network throughput and response time.

There are two interfaces to DECthreads. Routines prefixed with **cma** (for example, **cma_thread_create**) are part of the Concert Multithread Architecture, a stable, upwardly compatible interface to DECthreads. Routines prefixed with **pthread** (for example, **pthread_create**) comply with the POSIX 1003.4a draft standard for multithreading.

The following information is for your information only. It is not intended to be used as a basis for any action. The information is for your information only.

The following information is for your information only. It is not intended to be used as a basis for any action. The information is for your information only.

1st Overview

The following information is for your information only. It is not intended to be used as a basis for any action. The information is for your information only.

The following information is for your information only. It is not intended to be used as a basis for any action. The information is for your information only.

The following information is for your information only. It is not intended to be used as a basis for any action. The information is for your information only.

DECdtm System Services: New and Changed Features

The new features introduced in the DECdtm system services include:

- Support for reason codes on transaction abort
- Support for transaction timeouts
- New and modified System Dump Analyzer (SDA) Utility commands

Table 16-1 summarizes the DECdtm system services. For a detailed description of each system service, refer to the *VMS System Services Reference Manual*.

Table 16-1 DECdtm System Services Changes

System Service	Description	Comment
\$ABORT_TRANS	Abort Transaction	Supports new reason parameter and returns abort reason code in the I/O status block
\$ABORT_TRANSW	Abort Transaction and Wait	Supports new reason parameter and returns abort reason code in the I/O status block
\$END_TRANS	End Transaction	Returns abort reason code in the I/O status block if the transaction is aborted
\$END_TRANSW	End Transaction and Wait	Returns abort reason code in the I/O status block if the transaction is aborted
\$START_TRANS	Start Transaction	Supports new timeout and acmode parameters
\$START_TRANSW	Start Transaction and Wait	Supports new timeout and acmode parameters

16.1 Abort Reason Codes

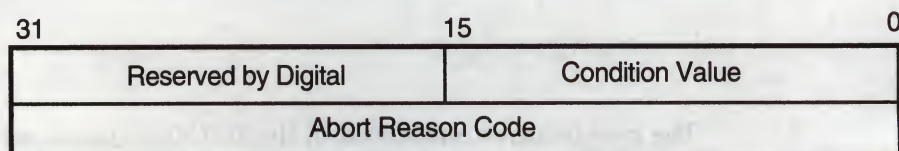
In order to better differentiate the causes of transaction failures, DECdtm services allow an abort reason code to be supplied when an application or resource manager aborts a transaction. When an application calls the \$ABORT_TRANS(W) system service to abort a transaction, it can supply an abort reason code in the **reason** parameter to specify why the transaction is to be aborted. Similarly, a resource manager that casts a "veto" vote may specify an abort reason code.

DECdtm System Services: New and Changed Features

16.1 Abort Reason Codes

The abort reason code is returned in the I/O status block (IOSB) for \$ABORT_TRANS(W) and \$END_TRANS(W). If multiple reasons are supplied by the application and resource managers, the DECdtm services will make an arbitrary decision about which abort reason code is returned in the IOSB. Figure 16-1 shows the structure of this IOSB.

Figure 16-1 IOSB Structure



ZK-3667A-GE

The abort reason codes are defined in the \$DDTMMSGDEF module. Refer to the description of \$ABORT_TRANS for the abort reason codes that can be used with or returned by the DECdtm system services.

16.2 Transaction Timeouts

With DECdtm services, it is possible to set a time limit for a given transaction. This value limits the amount of time the transaction may take to reach a commit decision. If this time limit is exceeded without the transaction being committed, the transaction is aborted. Applications may establish a timeout when calling the \$START_TRANS system service by using the **timeout** parameter.

16.3 New and Modified System Dump Analyzer Commands

The System Dump Analyzer (SDA) Utility has been modified to provide information about transactions and transaction log files. Table 16-2 summarizes the enhancements to the SDA commands. For complete reference information about these new and modified commands, see Chapter 20.

Table 16-2 SDA Utility Changes

SDA Command	Comment
New Commands	
SHOW LOGS	Displays information about transaction log files on the node
SHOW TRANSACTIONS	Displays information about transactions on the node
Modified Command	
SHOW PROCESS	New /TRANSACTIONS and /PARTICIPANTS qualifiers

LAT \$QIO Functions

This chapter describes the new LAT \$QIO functions SETMODE (IO\$_TTY_PORT!IO\$_M_LT_SETMODE) and SENSEMODE (IO\$_TTY_PORT!IO\$_M_LT_SENSEMODE).

17.1 LAT SETMODE \$QIO Function

The LAT SETMODE \$QIO function (IO\$_TTY_PORT!IO\$_M_LT_SETMODE) is used to create, delete, and modify LAT nodes, services, ports, and links.

The LAT SETMODE \$QIO function accepts four arguments: P1, P2, P3, and P4. P1 is the address of an item list; P2 is the length of this item list.

P3 specifies the type of entity to which the SETMODE operation applies. The entity type can be one of four types:

- LAT\$_C_ENT_NODE—Node. Only the local node name may be specified, with the exception of a SETMODE item list containing no item codes other than LAT\$_ITM_COUNTERS.
- LAT\$_C_ENT_SERVICE—Service. Only local service names may be specified, with the exception of a SETMODE item list containing no item codes other than LAT\$_ITM_COUNTERS.
- LAT\$_C_ENT_LINK—Link (the data link associated with Ethernet).
- LAT\$_C_ENT_PORT—Port.

The value for the entity type occupies the low-order 16 bits (bits 0–15) of the P3 parameter. For all four of the entity types, bits 16–19 are used as a status field to indicate the expected current status of the entity. These bits are used to decide whether the entity needs to be created before its characteristics are set. The possible values for this field are:

- LAT\$_C_ENTS_OLD—The entity must already exist. An SS\$_NOSUCHDEV error is returned if the entity does not exist.
- LAT\$_C_ENTS_NEW—The entity must be created. An SS\$_DUPLNAM error is returned if the entity already exists.
- LAT\$_C_ENTS_UNK—If the entity does not exist, it is created. If it does exist, its characteristics are modified.
- LAT\$_C_ENTS_DEL—If the entity exists, delete it. Otherwise, an SS\$_NOSUCHDEV error is returned and the item list is not used.

Creation, deletion, or modification of any entity requires the OPER privilege.

P4 may contain the address of an entity name string descriptor. If this parameter is omitted (contains a zero or the address of a descriptor that points to an empty buffer), a default may be used in some cases. The defaults for each entity type are as follows:

LAT \$QIO Functions

17.1 LAT SETMODE \$QIO Function

- LAT\$C_ENT_NODE—The local node.
- LAT\$C_ENT_SERVICE—No default; you must specify the service name.
- LAT\$C_ENT_LINK—The string "LAT\$LINK".
- LAT\$C_ENT_PORT—The device name associated with the currently assigned channel (the CHAN parameter of a \$QIO function).

Figure 17-1 shows an example of a SETMODE item list.

Figure 17-1 Example of SETMODE Item List

31		16 15		0	
LAT\$C_ON			LAT\$ITM_STATE		
LAT\$ITM_KEEPAIVE_TIMER					
40					
' L '	11		LAT\$ITM_IDENTIFICATION		
' C '	, ,		' C '	' T '	
' T '	' S '		' U '	' L '	
LAT\$ITM_CIRCUIT_TIMER			' R '	' E '	
160					
LAT\$C_ENABLED			LAT\$ITM_SERVER_MODE		
LAT\$ITM_USER_GROUPS					
13	0		4	5	
LAT\$OUTGOING_SES_LIMIT			9	1	
5					

ZK-3798A

This SETMODE item list is the P1 parameter for a \$QIO SETMODE function on the local node. P4 is omitted, and P3 is #LAT\$C_ENT_NODE!<LAT\$C_ENTS_OLD@16>. P2 is the length of the item list (52). A \$QIO SETMODE function for this item list would perform the following operations:

1. Set the state of the node to ON.
2. Set the LAT keepalive timer to 40 seconds.
3. Set the node identification to LTC CLUSTER.
4. Set the LAT circuit timer to 160 milliseconds.
5. Enable LAT outbound connections.
6. Turn on user groups 2, 8, 10, 11, 12, 16, and 19.
7. Set the outgoing session limit to 5 sessions.

SETMODE can return the following status codes:

- SS\$NOPRIV—No privilege to complete the desired operation.
- SS\$ACCVIO—Part of the argument list or item list is not addressable.

- **SS\$_BADPARAM**—One of the parameters in the item list is in error. If this value is returned, the second longword of the IOSB contains the item code of the parameter in error.

SETMODE Item Codes

Each item in the item list consists of a 1-word (16-bit) item code, followed by a value associated with the item.

Item codes in which the bit named **LAT\$V_STRING** is 0 take a longword value. The associated value is contained in the longword immediately following the item code in the item list. Item codes in which this bit is 1 take a counted string for their value. The byte immediately following the item code contains a byte count, which describes the length of the string that immediately follows it.

If you set bit **LAT\$V_CLEAR** in the item code to 1, the current value associated with the item code is cleared or set to its default value. In this case, the actual value specified in the item list is ignored, although the byte count field skips to the next item in the item list.

For each entity type, only a subset of item codes may be set. Table 17–1 lists the item codes that may be set for the **LAT\$C_ENT_NODE** entity type.

Table 17–1 LAT\$C_ENT_NODE Setmode Item Codes

Item Code	Meaning
LAT\$_ITM_STATE	Operating state of the LAT protocol. The following values are allowed: <div style="margin-left: 20px;">LAT\$_C_OFF Turn off LAT protocol processing. No new connections allowed in either direction. Existing connections are immediately terminated. This is the default.</div> <div style="margin-left: 20px;">LAT\$_C_SHUT Disallow new LAT connections in either direction. Existing connections are allowed to remain active.</div> <div style="margin-left: 20px;">LAT\$_C_ON Turn on LAT protocol processing.</div>
LAT\$_ITM_CIRCUIT_TIMER	Circuit timer value in milliseconds. Valid values are 10 to 1000 milliseconds. The default is 80 milliseconds.
LAT\$_ITM_CPU_RATING	CPU rating. Valid values are 0 to 100. If this value is 0, then the CPU rating value is not used in the rating calculation. See the <i>VMS LAT Control Program (LATCP) Manual</i> for a complete description of this feature.
LAT\$_ITM_DEVICE_SEED	Overrides the default lower boundary for new LTA devices. Valid values are 0 to 9999; the default is 0. See the <i>VMS LAT Control Program (LATCP) Manual</i> for a complete description of this feature.
LAT\$_ITM_KEEPA_LIVE_TIMER	Keepalive timer value in seconds. Valid values are 10 to 255 seconds. The default is 20 seconds.
LAT\$_ITM_MULTICAST_TIMER	Multicast timer value in seconds. Valid values are 10 to 180 seconds. The default is 60 seconds.
LAT\$_ITM_NODE_LIMIT	Maximum number of nodes in LAT database. The default is 0, where the maximum is determined by system resources.
LAT\$_ITM_RETRANSMIT_LIMIT	LAT retransmit limit. Valid values are 4 to 120 retransmissions. The default is 8 retransmissions.

(continued on next page)

LAT \$QIO Functions

17.1 LAT SETMODE \$QIO Function

Table 17-1 (Cont.) LAT\$C_ENT_NODE Setmode Item Codes

Item Code	Meaning
LAT\$_ITM_SERVER_MODE	Controls whether the node allows the use of the MASTER side of the LAT protocol for outbound connections. Possible values are: LAT\$C_DISABLED Server mode disabled (this is the default) LAT\$C_ENABLED Server mode enabled
LAT\$_ITM_SERVICE_RESPONDER	Indicates whether the node is to respond to service inquiries originating from a remote system. These inquiries are not necessarily directed at services being offered by the node. See the <i>VMS LAT Control Program (LATCP) Manual</i> for a complete description of this feature. Possible values are: LAT\$C_DISABLED Service inquiry response disabled (this is the default) LAT\$C_ENABLED Service inquiry response enabled
LAT\$_ITM_OUTGOING_SES_LIMIT	Maximum number of outgoing LAT sessions. A value of zero, which is the default, indicates that the limit is determined by system resources.
LAT\$_ITM_INCOMING_SES_LIMIT	Maximum number of interactive LAT sessions. A value of zero, which is the default, indicates that the limit is determined by system resources.
LAT\$_ITM_CONNECTIONS	Controls whether inbound connections can be accepted. Possible values are: LAT\$C_DISABLED Inbound connections disabled LAT\$C_ENABLED Inbound connections enabled (this is the default)
LAT\$_ITM_NODE_NAME	Causes the node LAT node name to be set to the given name. This item code may be specified only if the entity status field of the P3 parameter is LAT\$C_ENTS_NEW; otherwise, a LAT\$_ENTNOTFOU error results.
LAT\$_ITM_IDENTIFICATION	Node identification string. The default is the translation of SYS\$ANNOUNCE.
LAT\$_ITM_SERVICE_GROUPS	Specifies a default service group code bit mask. This mask is used when creating new local services. The default is group code 0 enabled and all others disabled when the LAT software is initialized.

Note

The use of the LAT\$V_CLEAR bit is an exception for this parameter code. If you clear bit LAT\$V_CLEAR, group codes corresponding to the group code mask, as specified in the item list, are set. Alternatively, if you set LAT\$V_CLEAR, group codes corresponding to the group code mask, as specified in the item list, will be cleared.

(continued on next page)

Table 17-1 (Cont.) LAT\$C_ENT_NODE Setmode Item Codes

Item Code	Meaning
LAT\$_ITM_USER_GROUPS	LAT group codes to be used when attempting outbound connections using the MASTER side of the LAT protocol. The default is all group codes disabled when the LAT software is initialized.
<p style="text-align: right;">Note</p> <p>The use of the LAT\$V_CLEAR bit is an exception for this parameter code. If you clear bit LAT\$V_CLEAR, group codes corresponding to the group code mask, as specified in the item list, will be set. Alternatively, if you set LAT\$V_CLEAR, group codes corresponding to the group code mask, as specified in the item list, are cleared.</p>	
LAT\$_ITM_COUNTERS	Node counters block. Allows for zeroing of all node counters. This item code may be specified only if the entity status field of the P3 parameter is LAT\$C_ENTS_OLD and the LAT\$V_CLEAR bit is set. Violating either of these two rules results in a returned status of SS\$_BADPARAM.
LAT\$_ITM_MAXIMUM_UNITS	Maximum unit number. Sets the highest value for a LTA unit number. Must be between 1 and 9999; defaults to 9999.

Table 17-2 lists the item codes that may be set for the LAT\$C_ENT_SERVICE entity type.

Table 17-2 LAT\$C_ENT_SERVICE Setmode Item Codes

Item Code	Meaning
LAT\$_ITM_RATING	Static LAT service rating. The default is the dynamic rating calculation. Static ratings can be between 0 and 255.
LAT\$_ITM_IDENTIFICATION	Service identification string. The default is the translation of SYS\$ANNOUNCE.
LAT\$_ITM_SERVICE_PASSWORD	Password string for locally offered service. The default is no password.
LAT\$_ITM_SERVICE_TYPE	Defines the type of service. Possible values are: LAT\$C_ST_GENERAL Creates a general timesharing service. LAT\$C_ST_APPLICATION Creates a special application service which must then be associated with ports dedicated to accepting connections to this service (dedicated ports)
LAT\$_ITM_COUNTERS	Service counters block. Allows for zeroing of all service counters. This item code may be specified only if the entity status field is LAT\$C_ENTS_OLD and the LAT\$V_CLEAR bit is set. Violating either of these two rules results in a returned status of SS\$_BADPARAM.

LAT \$QIO Functions

17.1 LAT SETMODE \$QIO Function

Table 17-3 lists the item codes that may be set for the LAT\$C_ENT_LINK entity type.

Table 17-3 LAT\$C_ENT_LINK Setmode Item Codes

Item Code	Meaning
LAT\$ITM_STATE	Operating state of the LAT protocol. Allowable values are:
LAT\$C_OFF	Turn off LAT protocol processing. No new connections allowed in either direction. Existing connections are immediately terminated.
LAT\$C_SHUT	Disallow new LAT connections in either direction. Existing connections are allowed to remain active.
LAT\$C_ON	Turn on LAT protocol processing. This is the default.
LAT\$ITM_DEVICE_NAME	The name of the Ethernet device to be used for this link. The default is hardware dependent.
LAT\$ITM_DECNET_ADDRESS	Specifies whether to use the DECnet address when starting the LAT protocol on the Ethernet controller associated with this link. Possible values are:
LAT\$C_DISABLED	DECnet address use disabled
LAT\$C_ENABLED	DECnet address use enabled (this is the default)
LAT\$ITM_COUNTERS	Link counters block. Allows for zeroing of all link counters. This item code may be specified only if the entity status field is LAT\$C_ENTS_OLD and the LAT\$V_CLEAR bit is set. Violating either of these two rules results in a returned status of SS\$BADPARAM.

Table 17-4 lists the item codes that may be set for the LAT\$C_ENT_PORT entity type.

Table 17-4 LAT\$C_ENT_PORT Setmode Item Codes

Item Code	Meaning
LAT\$ITM_PORT_TYPE	Type of port. Allowed values are:
LAT\$C_PT_APPLICATION	Application port for solicited connections.
LAT\$C_PT_DEDICATED	Dedicated port associated with a local application service.
LAT\$ITM_QUEUED	Controls whether the solicited connection requests queued or nonqueued access. Possible values are:
LAT\$C_DISABLED	Queued access disabled
LAT\$C_ENABLED	Queued access enabled (this is the default)

(continued on next page)

Table 17-4 (Cont.) LAT\$C_ENT_PORT Setmode Item Codes

Item Code	Meaning
LAT\$ ITM_SERVICE_ CLASS	Controls the class driver that the LAT driver communicates with when a connection is established. This item code can be used only with an entity status of NEW. Therefore, the service class must be specified when the device is created. An attempt to change the service class of an existing device returns SSS_BADPARAM. Legal values are: LAT\$C_SERVCLASS_ INTERACTIVE Service class 1, TTDRIVER (this is the default) LAT\$C_SERVCLASS_ TESTSERVICE Service class 2, TEST SERVICE LAT\$C_SERVCLASS_ XTRANSPORT Service class 3, X Protocol LAT\$C_SERVCLASS_FONT Service class 4, X fonts
LAT\$ ITM_DISPLAY_ NUMBER	For X devices, this is the binary value of the display number, which may need to be transmitted in some LAT messages. Values range from 0-255, with a default of 0. This item code has meaning only when used with service classes 3 and 4 (LAT\$C_SERVCLASS_XTRANSPORT AND LAT\$C_SERVCLASS_FONT).
LAT\$ ITM_TARGET_ NODE_NAME	Target node name for connection. This parameter must be specified for application ports and may optionally be specified for forward ports.
LAT\$ ITM_TARGET_ SERVICE_NAME	Target service name for connection. This parameter must be specified for forward ports and may optionally be specified for application ports. For dedicated ports, this parameter specifies the local application service to which the port should be associated.
LAT\$ ITM_TARGET_ PORT_NAME	Target port name for connection. This parameter may optionally be specified for application ports or forward ports; it is ignored for all other kinds of ports.
LAT\$ ITM_SERVICE_ PASSWORD	Password string for remote service on forward ports. This parameter must be specified to access services that are protected with a password. This parameter is ignored if it is specified for a service that is not protected with a password.

17.2 LAT SENSEMODE \$QIO Function

The LAT SENSEMODE \$QIO function (IO\$_TTY_PORT!IO\$_M_LT_SENSEMODE) is used to obtain information about LAT entities, including nodes, services, ports, and links.

The LAT SENSEMODE \$QIO function accepts four arguments: P1, P2, P3, and P4. P1 is the address of a buffer into which information about the desired entity is returned. The information is returned in the form of an item list. Unlike system services such as \$GETDVI or \$GETJPI, you do not select which items of information are returned. Information is returned in a fashion similar to the Ethernet device drivers' SENSEMODE operations. P2 is the length of the buffer specified in P1, in bytes. The number of bytes of information returned in the P1 buffer is returned in IOSB+2.

P3 specifies the type of entity to which the SENSEMODE operation applies. The entity type can be one of four types:

- LAT\$C_ENT_NODE—Node, including the local node
- LAT\$C_ENT_SERVICE—Service, including local services
- LAT\$C_ENT_LINK—Link (the data link associated with Ethernet)
- LAT\$C_ENT_PORT—Port

The value for the entity type occupies the low-order 16 bits (bits 0–15) of the P3 parameter. Bits 16–23 are used as a flag field. Two bits are currently defined within this field: LAT\$V_SENSE_NEXT and LAT\$V_SENSE_FULL. If the LAT\$V_SENSE_NEXT bit is 0, information about the current entity described by the P3 and P4 parameters is returned to the user; if this bit is 1, information about the next entity that logically follows the one described by P4 is returned. If LAT\$V_SENSE_FULL is 0, only those item codes marked SUMMARY in the following tables are returned; if this bit is 1, all item codes that describe the entity specified by the P3 and P4 parameters are returned.

P4 may contain the address of an entity name string descriptor. If this parameter is omitted (contains a zero or the address of a descriptor that points to an empty string) and the LAT\$V_SENSE_NEXT bit is set, information about the first entity that matches the entity type supplied by P3 is returned.

If P4 is omitted and the LAT\$V_SENSE_NEXT bit is 0, a default entity name may be used in some cases. The defaults for each entity type are as follows:

- LAT\$C_ENT_NODE—The local node.
- LAT\$C_ENT_SERVICE—No default; you must specify the service name.
- LAT\$C_ENT_LINK—The string "LAT\$LINK".
- LAT\$C_ENT_PORT—The device name associated with the currently assigned channel.

SENSEMODE can return the following failure return codes:

- SS\$_NOPRIV—No privilege to complete the desired operation.
- SS\$_ACCVIO—Part of the argument list or item list is not addressable.

SENSEMODE Item Codes

Each item in the item list starts with a 1-word (16-bit) item code that describes the type of information contained in the item. The item code is followed by a value associated with the item.

Item codes in which the bit named LAT\$V_STRING is 0 take a longword value. The associated value is contained in the longword immediately following the item code in the item list. Item codes in which this bit is 1 take a counted string for their value. The byte immediately following the item code contains a byte count, which describes the length of the string that immediately follows it.

Table 17–5 lists the item codes that are returned for the LAT\$C_ENT_NODE entity type. Item codes noted as LOCAL are returned only if the information being returned is for the local node. Item codes noted as REMOTE are returned only if the information being returned is for a remote node. Item codes noted as BOTH are returned for both types of nodes.

Table 17–5 LAT\$C_ENT_NODE Sensemode Item Codes

Item Codes	Meaning
LAT\$ ITM_NODE_NAME (BOTH, SUMMARY)	LAT node name for the node.

(continued on next page)

Table 17-5 (Cont.) LAT\$C_ENT_NODE Sensemode Item Codes

Item Codes	Meaning
LAT\$_ITM_IDENTIFICATION (BOTH, SUMMARY)	Node identification string.
LAT\$_ITM_NODE_TYPE (BOTH, SUMMARY)	Type of node. Possible values are: LAT\$C_NT_LOCAL Node is local node. LAT\$C_NT_REMOTE Node is remote node.
LAT\$_ITM_STATE (LOCAL, SUMMARY)	Operating state of the LAT protocol. Possible values are: LAT\$C_ON New connections are allowed and the LAT protocol is running. LAT\$C_OFF New connections are not allowed. The LAT protocol is not running. LAT\$C_SHUT No new connections are allowed. Currently active connections are still maintained. The LAT protocol remains running only until the last active session is disconnected, at which time the node is placed in the OFF state.
LAT\$_ITM_NODE_STATUS (REMOTE, SUMMARY)	Current status of remote node. This item code is present only if a LAT virtual circuit does not currently exist between the local node and this remote node. Possible values are: LAT\$C_REACHABLE Remote node is reachable. LAT\$C_UNREACHABLE Remote node is unreachable. LAT\$C_UNKNOWN Remote node status is unknown.
LAT\$_ITM_CONNECTED_COUNT (REMOTE, SUMMARY)	Number of LAT sessions from the local node to this remote node. This item code replaces the LAT\$_ITM_NODE_STATUS item code for remote nodes to which a LAT virtual circuit currently exists.
LAT\$_ITM_SERVICE_GROUPS (BOTH)	Bit mask of LAT group codes that are serviced by the node.
LAT\$_ITM_PROTOCOL_VERSION (BOTH)	LAT protocol version string.
LAT\$_ITM_DATA LINK_ADDRESS (REMOTE)	Ethernet address that is used by the (remote) node.
LAT\$_ITM_NODE_LIMIT	Maximum number of nodes in LAT database. The default is zero; the maximum is determined by system resources.
LAT\$_ITM_RETRANSMIT_LIMIT	LAT retransmit limit. Valid values are 1 to 255 retransmissions. The default is 20 retransmissions.
LAT\$_ITM_MAXIMUM_UNITS (LOCAL)	Maximum LTA unit number.
LAT\$_ITM_SERVER_MODE (LOCAL)	Controls whether the node allows the use of the MASTER side of the LAT protocol for outbound connections. Possible values are: LAT\$C_DISABLED Server mode disabled (this is the default) LAT\$C_ENABLED Server mode enabled

(continued on next page)

LAT \$QIO Functions

17.2 LAT SENSEMODE \$QIO Function

Table 17-5 (Cont.) LAT\$C_ENT_NODE Sensemode Item Codes

Item Codes	Meaning
LAT\$ _ITM_SERVICE_RESPONDER (LOCAL)	Indicates whether the node is to respond to service inquiries originating from a remote system. These inquiries are not necessarily directed at services being offered by the node. See the <i>VMS LAT Control Program (LATCP) Manual</i> for a complete description of this feature. Possible values are: LAT\$C_DISABLED Service inquiry response disabled (this is the default) LAT\$C_ENABLED Service inquiry response enabled
LAT\$ _ITM_OUTGOING_SES_LIMIT (LOCAL)	Maximum number of outgoing LAT sessions. A value of zero indicates no limit. The default is zero; the maximum is determined by system resources.
LAT\$ _ITM_INCOMING_SES_LIMIT (LOCAL)	Maximum number of interactive LAT sessions. A value of zero indicates no limit. The default is zero; the maximum is determined by system resources.
LAT\$ _ITM_USER_GROUPS (LOCAL)	Bit mask of LAT group codes to be used when attempting outbound connections using the MASTER side of the LAT protocol.
LAT\$ _ITM_CIRCUIT_TIMER (BOTH)	Circuit timer value in milliseconds. The default is zero.
LAT\$ _ITM_CPU_RATING (LOCAL)	CPU rating.
LAT\$ _ITM_KEEPA_LIVE_TIMER (LOCAL)	Keepalive timer in seconds. The default is 20.
LAT\$ _ITM_MULTICAST_TIMER (BOTH)	Multicast timer value in seconds. The default is 20.
LAT\$ _ITM_CONNECTIONS (BOTH)	Indicates whether inbound connections (interactive sessions) can be accepted. Possible values are: LAT\$C_DISABLED Service inquiry response disabled LAT\$C_ENABLED Service inquiry response enabled (this is the default)

Node service information is presented as a list of node service subblocks, with each subblock containing information about one particular service offered by the node. The subblock item code LAT\$ _ITM_NODE_SVC_BLOCK has the LAT\$V_STRING bit set to 1, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an item list and consists of the item codes listed in Table 17-6.

Table 17-6 Node Service Subblock Item Codes

LAT\$ _ITM_SERVICE_NAME (BOTH)	Name of a LAT service offered by the node.
LAT\$ _ITM_SERVICE_STATUS (BOTH)	Status of the service. Possible values are LAT\$C_AVAILABLE and LAT\$C_UNAVAILABLE.
LAT\$ _ITM_SERVICE_TYPE (LOCAL)	Type of service. Possible values are LAT\$C_ST_GENERAL (general timesharing service) and LAT\$C_ST_APPLICATION (special application service associated with ports dedicated to accepting connections to this service).

(continued on next page)

Table 17-6 (Cont.) Node Service Subblock Item Codes

LAT\$ _ITM_RATING (BOTH)	LAT service rating associated with the service.
LAT\$ _ITM_RATING_TYPE (LOCAL)	Type of LAT rating calculation being done by this node. Possible values are LAT\$C_STATIC and LAT\$C_DYNAMIC.
LAT\$ _ITM_IDENTIFICATION (BOTH)	Identification string associated with the service.

Node counters information is presented as a counters subblock. The subblock item code LAT\$ _ITM_COUNTERS has the LAT\$V_STRING bit set to 1, and the string length byte actually contains the length of the entire subblock. The subblock itself is an item list and consists of the item codes listed in Table 17-7.

Table 17-7 Node Counters Item Codes

LAT\$ _ITM_CTNOd_SSZ (BOTH)	Seconds since zeroed
LAT\$ _ITM_CTNOd_MSGr (BOTH)	Messages received
LAT\$ _ITM_CTNOd_MSGT (BOTH)	Messages transmitted
LAT\$ _ITM_CTNOd_SLTR (BOTH)	Slots received
LAT\$ _ITM_CTNOd_SLTT (BOTH)	Slots transmitted
LAT\$ _ITM_CTNOd_BYTR (BOTH)	Bytes received
LAT\$ _ITM_CTNOd_BYTT (BOTH)	Bytes transmitted
LAT\$ _ITM_CTNOd_MNA (BOTH)	Multiple node addresses
LAT\$ _ITM_CTNOd_DUP (BOTH)	Duplicates received
LAT\$ _ITM_CTNOd_MRT (BOTH)	Messages retransmitted
LAT\$ _ITM_CTNOd_ILM (BOTH)	Illegal messages received
LAT\$ _ITM_CTNOd_ILS (BOTH)	Illegal slots received
LAT\$ _ITM_CTNOd_SLCA (BOTH)	Solicitations accepted
LAT\$ _ITM_CTNOd_SLCR (BOTH)	Solicitations rejected
LAT\$ _ITM_CTNOd_TER (LOCAL)	Transmit errors
LAT\$ _ITM_CTNOd_RES (LOCAL)	Resource errors
LAT\$ _ITM_CTNOd_NTb (LOCAL)	No transmit buffer

(continued on next page)

LAT \$QIO Functions

17.2 LAT SENSEMODE \$QIO Function

Table 17-7 (Cont.) Node Counters Item Codes

LAT\$_ITM_CTNOB_TMO (LOCAL)	Virtual circuit timeout
LAT\$_ITM_CTNOB_DOB (LOCAL)	Discarded output bytes
LAT\$_ITM_CTNOB_LSTER (LOCAL)	Last transmit error
LAT\$_ITM_CTNOB_MCBXMT (LOCAL)	Number of multicast bytes transmitted
LAT\$_ITM_CTNOB_MCBRCV (LOCAL)	Number of multicast bytes received
LAT\$_ITM_CTNOB_MCMXMT (LOCAL)	Number of multicast messages transmitted
LAT\$_ITM_CTNOB_MCMRCV (LOCAL)	Number of multicast messages received
LAT\$_ITM_CTNOB_SOLFAIL (LOCAL)	Number of solicitation failures
LAT\$_ITM_CTNOB_ATLOS (LOCAL)	Number of times attention slot data was lost
LAT\$_ITM_CTNOB_DATLOS (LOCAL)	Number of times user data was lost
LAT\$_ITM_CTNOB_NOREJ (LOCAL)	Number of time a reject slot could not be sent
LAT\$_ITM_CTNOB_LOSCT (LOCAL)	Number of times remote node counters were lost
LAT\$_ITM_CTNOB_LOSSAM (LOCAL)	Number of service announcement messages lost
LAT\$_ITM_CTNOB_NOSAM (LOCAL)	Number of times a service announcement message could not be sent
LAT\$_ITM_CTNOB_NOSTS (LOCAL)	Number of times node status was lost
LAT\$_ITM_CTNOB_NOXMT (LOCAL)	Number of times no link was available for a transmit
LAT\$_ITM_CTNOB_CTLERR(LOCAL)	Number of controller errors
LAT\$_ITM_CTNOB_CERRCOD(LOCAL)	Lost controller error
LAT\$_ITM_CTNOB_ISOLA(LOCAL)	Number of incoming solicitations accepted
LAT\$_ITM_CTNOB_ISOLR(LOCAL)	Number of incoming solicitations rejected
LAT\$_ITM_CTNOB_PROTO (LOCAL)	Protocol error count

Several protocol errors are also included in a separate subblock. The protocol errors item code is LAT\$_ITM_PROTOCOL_ERRORS and has LAT\$V_STRING set (the size of the subblock is contained in the first byte following the item code). The item codes and the events they represent are listed in Table 17-8.

Table 17-8 Protocol Error Item Codes

Item Codes	Meaning
LAT\$ _ITM_CTPRO_IVM (LOCAL)	Invalid message type received
LAT\$ _ITM_CTPRO_ISM (LOCAL)	Invalid start message received
LAT\$ _ITM_CTPRO_IVS (LOCAL)	Invalid sequence number received
LAT\$ _ITM_CTPRO_NIZ (LOCAL)	Zero-node index received
LAT\$ _ITM_CTPRO_ICI (LOCAL)	Node circuit index out of range
LAT\$ _ITM_CTPRO_CSI (LOCAL)	Node circuit sequence invalid
LAT\$ _ITM_CTPRO_NLV (LOCAL)	Node circuit index no longer valid
LAT\$ _ITM_CTPRO_HALT (LOCAL)	Circuit was forced to halt
LAT\$ _ITM_CTPRO_MIZ (LOCAL)	Invalid master slot index
LAT\$ _ITM_CTPRO_SIZ (LOCAL)	Invalid slave slot index
LAT\$ _ITM_CTPRO_CRED (LOCAL)	Invalid credit field
LAT\$ _ITM_CTPRO_RCSM (LOCAL)	Repeat creation of slot by master
LAT\$ _ITM_CTPRO_RDSM (LOCAL)	Repeat disconnection of slot by master

Table 17-9 lists the item codes that are returned for the LAT\$C_ENT_SERVICE entity type. As in Table 17-5, item codes noted as LOCAL are returned only if the information being returned is for a locally offered service. Item codes noted as REMOTE are returned only if the information being returned is for a service offered by a remote node. Item codes noted as BOTH are returned for both types of services.

Table 17-9 LAT\$C_ENT_SERVICE Sensemode Item Codes

Item Codes	Meaning
LAT\$ _ITM_SERVICE_NAME (BOTH, SUMMARY)	Service name.
LAT\$ _ITM_SERVICE_STATUS (BOTH, SUMMARY)	Status of the specified service. Possible values are LAT\$C_AVAILABLE and LAT\$C_UNAVAILABLE.
LAT\$ _ITM_SERVICE_TYPE (LOCAL,SUMMARY)	Type of service. Possible values are LAT\$C_ST_GENERAL (general timesharing service) and LAT\$C_ST_APPLICATION (special application service associated with ports dedicated to accepting connections to this service).

(continued on next page)

LAT \$QIO Functions

17.2 LAT SENSEMODE \$QIO Function

Table 17–9 (Cont.) LAT\$C_ENT_SERVICE Sensemode Item Codes

Item Codes	Meaning
LAT\$ _ITM_IDENTIFICATION (BOTH, SUMMARY)	Service identification string, as advertised by the highest rated node that currently offers the service.

Service node information is presented as a list of service node subblocks, with each subblock containing information about one particular node that offers the service. The subblock item code LAT\$ _ITM_SVC_NODE_BLOCK has the LAT\$V_STRING bit set to 1, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an item list and consists of the item codes listed in Table 17–10.

Table 17–10 Service Node Subblock Item Codes

LAT\$ _ITM_NODE_NAME (BOTH)	Name of a LAT node that offers the selected service.
LAT\$ _ITM_STATE (LOCAL)	Current state of the LAT protocol on the local node. Possible values are: LAT\$C_ON—New connections are allowed, and the LAT protocol is running. LAT\$C_OFF—New connections are not allowed, and any current connections are abnormally terminated. The LAT protocol is not running. LAT\$C_SHUT—No new connections are allowed. Currently active connections are still maintained. The LAT protocol remains running only until the last active session is disconnected, at which time the node is placed in the OFF state.
LAT\$ _ITM_NODE_STATUS (REMOTE)	Current status of the remote node. This item code is present only if a LAT virtual circuit does not currently exist to the remote node. Possible values are LAT\$C_REACHABLE, LAT\$C_UNREACHABLE, and LAT\$C_UNKNOWN.
LAT\$ _ITM_CONNECTED_COUNT (REMOTE)	Number of LAT sessions from the local node to this remote node. This item code replaces the LAT\$ _ITM_NODE_STATUS item code for remote nodes to which a LAT virtual circuit currently exists.
LAT\$ _ITM_RATING (BOTH)	LAT service rating associated with the service.
LAT\$ _ITM_RATING_TYPE (LOCAL)	Type of LAT rating calculation being done by this node. Possible values are LAT\$C_STATIC and LAT\$C_DYNAMIC.
LAT\$ _ITM_IDENTIFICATION (BOTH)	Identification string associated with the service.

Service counters information is presented as a counters subblock. The subblock item code LAT\$ _ITM_COUNTERS has the LAT\$V_STRING bit set, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an item list and consists of the item codes listed in Table 17–11.

Table 17–11 Service Counters Subblock Item Codes

LAT\$ _ITM_CTSRV_SSZ (BOTH)	Seconds since zeroed.
LAT\$ _ITM_CTSRV_MCNA (BOTH)	Outgoing connections attempted (the number of times the local node has attempted to connect to the service offered on a remote node).

(continued on next page)

Table 17-11 (Cont.) Service Counters Subblock Item Codes

LAT\$ _ITM_CTSRV_MCNC (BOTH)	Outgoing connections completed (the number of times the local node successfully connected to the service offered on a remote node).
LAT\$ _ITM_CTSRV_SCNA (BOTH)	Incoming connections accepted (the number of times the local node has accepted a connection request from a remote node to the locally offered service).
LAT\$ _ITM_CTSRV_SCNR (BOTH)	Incoming connections rejected (the number of times the local node rejected a connection request from a remote node to the locally offered service).
LAT\$ _ITM_DED_PORT_ BLOCK (LOCAL)	If the selected service is an application service offered by the local node, a list of one or more port subblocks is included in the item list. These subblocks describe the dedicated port or ports associated with this application service, with each subblock describing one particular port. The subblock item code LAT\$ _ITM_DED_PORT_BLOCK has the LAT\$V_STRING bit set, and the string length byte actually contains the length of the entire subblock. Each subblock itself is an item list and currently consists only of the following item code: LAT\$ _ITM_PORT_NAME Name of the dedicated port (LOCAL)

Table 17-12 lists the item codes that are returned for the LAT\$C_ENT_LINK entity type.

Table 17-12 LAT\$C_ENT_LINK Sensemode Item Codes

Item Codes	Meaning
LAT\$ _ITM_LINK_ NAME (SUMMARY)	Link name (such as LAT\$LINK).
LAT\$ _ITM_STATE_ SUMMARY	State of the link. Possible values are: LAT\$C_ON New connections are allowed, and the LAT protocol is running. LAT\$C_OFF New connections are not allowed, and any current connections are abnormally terminated. The LAT protocol is not running. LAT\$C_SHUT No new connections are allowed. Currently active connections are still maintained. The LAT protocol remains running only until the last active session is disconnected, at which time the node is placed in the OFF state.
LAT\$ _ITM_DEVICE_ NAME SUMMARY	The name of the Ethernet device used for the link.
LAT\$ _ITM_DATALINK_ ADDRESS	The Ethernet device's current physical address for the link.
LAT\$ _ITM_DECNET_ ADDRESS	Indicates whether the link attempts to use the default DECnet Ethernet address when starting the data link controller (enabling the LAT protocol). Possible values are: LAT\$C_DISABLED DECnet Ethernet address use disabled LAT\$C_ENABLED DECnet Ethernet address use enabled (this is the default)

Link counters information is presented as a counters subblock. The subblock item code LAT\$ _ITM_COUNTERS has the LAT\$V_STRING bit set, and the string length byte actually contains the length of the entire subblock. Because the link counters are independent of the protocol type, they include not only LAT

LAT \$QIO Functions

17.2 LAT SENSEMODE \$QIO Function

messages and events, but also all other protocol messages and events (that is, DECnet) associated with the same Ethernet device. The counters are actually maintained by the Ethernet device driver and are identified within the subblock by the non-protocol-specific item codes listed in Table 17-13.

Table 17-13 Link Counters Item Codes

NMA\$C_CTLIN_ZER	Seconds since zeroed
NMA\$C_CTLIN_DBR	Messages received
NMA\$C_CTLIN_DBS	Messages transmitted
NMA\$C_CTLIN_MBL	Multicast messages received
NMA\$C_CTLIN_MBS	Multicast messages transmitted
NMA\$C_CTLIN_BRC	Bytes received
NMA\$C_CTLIN_BSN	Bytes transmitted
NMA\$C_CTLIN_MBY	Multicast bytes received
NMA\$C_CTLIN_MSN	Multicast bytes transmitted
NMA\$C_CTLIN_RFL	Receive errors
NMA\$C_CTLIN_SFL	Transmit errors
NMA\$C_CTLIN_OVR	Data overrun
NMA\$C_CTLIN_UBU	User buffer unavailable
NMA\$C_CTLIN_SBU	System buffer unavailable
NMA\$C_CTLIN_LBE	Local buffer errors
NMA\$C_CTLIN_BS1	Messages sent, single collisions
NMA\$C_CTLIN_BSM	Messages sent, multiple collisions
NMA\$C_CTLIN_BID	Messages sent, initially deferred
NMA\$C_CTLIN_CDC	Transmit collision detection check failure

Table 17-14 lists the item codes that are returned for the LAT\$C_ENT_PORT entity type.

Table 17-14 LAT\$C_ENT_PORT Sensemode Item Codes

Item Codes	Meaning
LAT\$_ITM_PORT_NAME SUMMARY	Name of the port (such as _LTA15:).
LAT\$_ITM_PORT_TYPE SUMMARY	Type of port. Possible values are:
LAT\$C_PT_FORWARD	Forward port used for outgoing LAT connections or for management functions
LAT\$C_PT_INTERACTIVE	Interactive port created as the result of an incoming LAT connection request
LAT\$C_PT_APPLICATION	Application port for solicited connections
LAT\$C_PT_DEDICATED	Dedicated port associated with a local service

(continued on next page)

Table 17-14 (Cont.) LAT\$C_ENT_PORT Sensemode Item Codes

Item Codes	Meaning
LAT\$_ITM_QUEUED	Controls whether the solicited connection requests queued or nonqueued access. Possible values are: LAT\$C_DISABLED Queued access disabled LAT\$C_ENABLED Queued access enabled (this is the default)
LAT\$_ITM_SERVICE_CLASS	Controls the class driver that the LAT driver communicates with when a connection is established. This item code can be used only with an entity status of NEW. Therefore, the service class must be specified when the device is created. An attempt to change the service class of an existing device returns SS\$_BADPARAM. Legal values are: LAT\$C_SERVCLASS_INTERACTIVE Service class 1, TTDRIVER (this is the default) LAT\$C_SERVCLASS_TESTSERVICE Service class 2, TEST SERVICE LAT\$C_SERVCLASS_XTRANSPORT Service class 3, X Protocol LAT\$C_SERVCLASS_FONT Service class 4, X fonts
LAT\$_ITM_DISPLAY_NUMBER	Display number value for the device. This field has meaning for service classes 3 and 4 only. It returns a value of zero for all other service classes.
LAT\$_ITM_DISCONNECT_REASON	Reason (if any) for the last disconnect on the port.

Note

The following four item codes are returned only when the LTA port has an active LAT connection.

LAT\$_ITM_CONNECTED_SERVICE_NAME	Name of service to which this port is connected. For forward and application ports, this is the name of the remote service to which the port is connected (if any). For interactive and dedicated ports, this is the name of the local service that accepted the remote-initiated connection.
LAT\$_ITM_CONNECTED_NODE_NAME	Name of remote node to which this port is connected.
LAT\$_ITM_CONNECTED_PORT_NAME	Name of remote port to which this port is connected.
LAT\$_ITM_CONNECTED_LINK_NAME	Name of the link that the LAT connection exists on.

Note

The following three items show information about how the port is set up. These items may be returned even if there is no current LAT connection.

(continued on next page)

LAT \$QIO Functions

17.2 LAT SENSEMODE \$QIO Function

Table 17-14 (Cont.) LAT\$C_ENT_PORT Sensemode Item Codes

Item Codes	Meaning
LAT\$_ITM_TARGET_SERVICE_NAME	Target service name for connection of forward or application ports. For dedicated ports, this item code specifies the local service with which the port is associated.
LAT\$_ITM_TARGET_NODE_NAME	Target node name for connection of forward or application ports.
LAT\$_ITM_TARGET_PORT_NAME	Target port name for connection of forward or application ports.

Asynchronous Printer Support

A new `TT$M_COMMSYNC` terminal characteristic has been added to the terminal driver interface and a new `/COMMSYNC` qualifier has been added to the DCL command `SET TERMINAL`. Both enable an asynchronous printer to be connected to a terminal port, with standard EIA modem control signals used for flow control.

A description of the `SET TERMINAL/COMMSYNC` command follows.

SET TERMINAL/COMMSYNC/NOCOMMSYNC

SET TERMINAL/COMMSYNC/NOCOMMSYNC

Allows asynchronous printers and other devices to be connected to terminal ports.

Format

/COMMSYNC
/NOCOMMSYNC (default)

Description

The /COMMSYNC qualifier allows asynchronous printers and other devices to be connected to terminal ports. When you specify /COMMSYNC, flow control is handled by standard EIA modem signals instead of by XON/XOFF.

Specifying /COMMSYNC activates the data terminal ready (DTR) and request to send (RTS) signals. Data is sent once the data set ready (DSR) and clear to send (CTS) signals are also present. If either of these signals is not present, printing stops. When both signals are present again, printing resumes.

Do not set the /COMMSYNC qualifier on a line with a modem hooked up on it that is intended for interactive use. The qualifier disables the modem terminal characteristic that disconnects a user process from the terminal line in case of a modem phone line failure. With the /COMMSYNC qualifier enabled, the next call on the terminal line could be attached to the previous user's process. /COMMSYNC should also not be used in combination with XON/XOFF (this can result in a hung state, even though nothing appears wrong) or in combination with /TTYSYNC or /HOSTSYNC (this complicates troubleshooting). The /COMMSYNC and /MODEM qualifiers are mutually exclusive.

Security administrators should be aware that /COMMSYNC should *not* be used on interactive terminal ports or on a port connected to a LAT line.

Third-party drivers that are used in conjunction with the VMS terminal driver (TTDRIVER) must be recompiled and relinked in order to use SET TERMINAL /COMMSYNC.

Example

```
$ SET TERMINAL/COMMSYNC
```

In this example, the SET TERMINAL command enables an asynchronous printer to be connected to the current terminal port.

Support for Case Sensitivity

The VMS linker and the MACRO assembler now support case sensitivity. Case sensitivity is the capability to sense and act upon alphabetic input with regard to its being uppercase or lowercase.

19.1 Linker Support for Case-Sensitive Languages

The VMS Linker Utility, with VMS Version 5.5, implements a new linker option, `CASE_SENSITIVE=`, that allows you to preserve the mixture of uppercase and lowercase characters used in character-string arguments to linker options. When this option is enabled, the linker interprets the symbols *MySymbol* and *mysymbol* as two distinct character strings. Once case sensitivity has been enabled, the linker preserves the case of all succeeding character-string arguments to linker options until you explicitly disable it. When the `CASE_SENSITIVE=` option is disabled (which is the default), the linker changes all the characters in a character string to uppercase before processing the string.

Note that the `CASE_SENSITIVE=` option only affects how the linker processes arguments to linker options. When it searches object files and shareable image files for symbols that need to be resolved, the linker preserves the case used in the symbol names (created by the language compilers). Also, the names of the linker options (all the characters preceding the equal sign [=], as in the `NAME=` option) are unaffected by the case-sensitivity option. The linker changes all the characters in option names to uppercase characters before processing the option, even if case sensitivity has been enabled.

To enable case sensitivity, specify the `CASE_SENSITIVE=` option with the value `YES` on a line in the options file. (You can specify only one option per line in a linker option file.) You can use any mixture of uppercase and lowercase characters in `YES`.

To disable case sensitivity, specify the `CASE_SENSITIVE=` option with the value `NO` as its argument. Note that, because case sensitivity is enabled, you must use uppercase characters when specifying `NO`.

Example 19-1 illustrates how to use this linker option.

Support for Case Sensitivity

19.1 Linker Support for Case-Sensitive Languages

Example 19-1 Using the CASE_SENSITIVE= Option

```
$ link/share/map/full test, sys$input:/opt ❶
case_sensitive=YES ❷
name=ImageName ❸
symbol=OneSymbol,1
case_sensitive=NO ❹
universal=myroutine ❺
Ctrl/z
```

The following list explains how the CASE_SENSITIVE= option is used in Example 19-1:

- ❶ By specifying the logical name SYS\$INPUT: as the linker option file, you can specify linker options at the command line.
- ❷ Specifying the CASE_SENSITIVE= option with YES enables case sensitivity in the linker options file.
- ❸ Because case sensitivity has been enabled, the linker preserves the mix of uppercase and lowercase characters used in character-string arguments to all succeeding linker options. In the example, this includes the character string *ImageName* passed to the NAME= option and the character string *OneSymbol* passed to the SYMBOL= option.
- ❹ Specifying the CASE_SENSITIVE= option with NO turns off case sensitivity. Note that you must use uppercase characters when specifying NO.
- ❺ Because case sensitivity has been disabled, the linker changes all the characters in the universal symbol, *myroutine*, to uppercase. The following excerpt from the map file produced by this link command illustrates how these identifiers were stored by the linker:

```
ImageName
OneSymbol
MYROUTINE
```

Carefully delimit the section of a linker options file in which you use case sensitivity to avoid unintentional side effects. For example, if you include options in the case-sensitive region that accept values such as YES, NO, EXE, and SHR, make sure the values are specified using uppercase characters. Because these values appear after the equal sign (=), they are affected by case sensitivity. Similarly, character-string arguments used to name a psect, cluster, or image are also affected by case sensitivity.

19.2 VAX MACRO Support for Case Sensitivity

VAX MACRO now enables programmers to specify the case sensitivity of global symbol definitions. This is accomplished using the new MACRO command line qualifier, /NAMES.

```
MACRO/NAMES = { UPPER          } ! Symbol definitions in uppercase (default)
               { DEFINITIONS:LOWER } ! Symbol definitions in lowercase
               { DEFINITIONS:UPPER } ! Symbol definitions in uppercase
               { DEFINITIONS:BOTH  } ! Symbol definitions in both
                                   ! uppercase and lowercase
```

The /NAMES qualifier enables you to observe case sensitivity when referencing MACRO routines in languages that generate references in lowercase.

`/NAMES=UPPER` specifies that all global symbol definitions are converted and generated in all lowercase characters. This is the default case and is consistent with the behavior of the current product.

If you specify `/NAMES=DEFINITIONS:LOWER`, then all global symbol definitions are converted and generated in all lowercase characters. If you specify `/NAMES=DEFINITIONS:UPPER`, then all global symbol definitions are converted and generated in all uppercase characters. There is no mixed casing or the ability to select which symbols within a module are generated in uppercase and which are generated in lowercase.

The `/NAMES=DEFINITIONS:BOTH` option generates the symbol definitions in both uppercase and lowercase.

This is a positional qualifier; therefore, you can specify which modules are affected by the qualifier.

Only the global symbol definitions are generated in the specified case sensitivity. The generation of requests, that is, calls or jumps to subroutines (JSBs), remain unchanged (uppercase).

Examples

1. `$ MACRO/NAMES=DEFINITIONS:UPPER TEST.MAR`

In this example, global symbol definitions from TEST.MAR are produced in the resulting object file (TEST.OBJ) in uppercase.

2. `$ MACRO TEST.MAR/NAMES=UPPER, TEST2.MAR/NAMES=DEFINITIONS:LOWER`

In this example, global symbol definitions from TEST.MAR are produced in the resulting object file (TEST.OBJ) in uppercase and global symbol definitions from TEST2.MAR are produced in the resulting object file (TEST2.OBJ) in lowercase.

Sections 19.2.1 through 19.2.4 describe how the `/NAMES` qualifier functions in different environments.

19.2.1 MACRO Programs That Reference Other MACRO Modules

A MACRO program can reference global symbols only in other MACRO modules that have been assembled using either the UPPER or BOTH case-sensitivity value. This also implies that modules that define transfer vectors and are referenced by separate MACRO modules follow this rule.

In the following table, the transfer vector module consists of transfer vector definitions only. The caller modules are assembled using the UPPER case-sensitivity value.

Caller	Called Routine (In Separate Module)
MACRO	MACRO ¹
MACRO	Transfer Vector ¹

¹Modules must be assembled using either the UPPER or BOTH case-sensitivity value.

19.2.2 MACRO Programs That Reference the Same MACRO Module

MACRO routines that reference global symbols that are defined in the same module, but cross program sections, must be assembled using either the UPPER or BOTH case-sensitivity value. This means that a module consisting of both transfer vector definitions and code in independent PSECTs follow the same rule.

In the following table, the MACRO module contains both the caller and called routine, but they reside in separate PSECTs. These modules must be assembled using either the UPPER or BOTH case-sensitivity value.

Caller	Called Routine (In Same Module)
MACRO	MACRO ¹
Transfer Vector	MACRO ¹

¹Modules must be assembled using either the UPPER or BOTH case-sensitivity value.

19.2.3 Uppercase Languages to MACRO Programs

MACRO modules that define transfer vectors must be assembled using either the UPPER or BOTH case-sensitivity value when used by MACRO or any other uppercase language. This imposes the same case sensitivity on the called routine.

Caller	Transfer Module	Called Routine
MACRO or any other uppercase language	MACRO/UPPER	MACRO/UPPER or BOTH
	MACRO/BOTH	MACRO/BOTH (required)
	MACRO/UPPER	Any mixed-case language, for example, C. Routine name must be in uppercase.

19.2.4 Lowercase Languages to MACRO Programs

Languages other than MACRO can use MACRO transfer vectors by means of a LOWER request (as long as they support generating lowercase requests). This requires that the MACRO module that contains the transfer directive definitions must be assembled using the BOTH case-sensitivity value. If the routine is in MACRO, then this module must also be assembled using the BOTH case-sensitivity value. Other languages that are referenced by the transfer vector must generate the symbol definition in both uppercase and lowercase. This is necessary to allow the linker the ability to resolve the symbolic references.

Caller	Transfer Module	Called Routine
Lowercase call	MACRO/BOTH	MACRO/BOTH
	MACRO/BOTH	Any mixed-case language, for example, C. Symbol definition name must be generated in uppercase and lowercase.

19.2.4.1 MACRO Command /NAMES Qualifier

This section describes the /NAMES qualifier for the MACRO command.

MACRO/NAMES

Allows you to specify the case sensitivity of global symbol definitions.

Format

MACRO filespec[,...]

Description

Starting with VMS Version 5.5, the DCL command MACRO accepts a new qualifier, /NAMES. The /NAMES qualifier enables you to observe case sensitivity when referencing MACRO routines in languages that generate references in lowercase.

/NAMES=UPPER specifies that all global symbol definitions are converted and generated in all lowercase characters. This is the default case and is consistent with the behavior of the current product.

If you specify /NAMES=DEFINITIONS:LOWER, then all global symbol definitions are converted and generated in all lowercase characters. If you specify /NAMES=DEFINITIONS:UPPER, then all global symbol definitions are converted and generated in all uppercase characters. There is no mixed casing or the ability to select which symbols within a module are generated in uppercase and which are generated in lowercase.

The /NAMES=DEFINITIONS:BOTH option generates the symbol definitions in both uppercase and lowercase.

This is a positional qualifier; therefore, you can specify which modules are affected by the qualifier.

Only the global symbol definitions are generated in the specified case sensitivity. The generation of requests, that is, calls or jumps to subroutines (JSBs), remain unchanged (uppercase).

Examples

1. \$ MACRO/NAMES=DEFINITIONS:UPPER TEST.MAR

In this example, global symbol definitions from TEST.MAR are produced in the resulting object file (TEST.OBJ) in uppercase.

2. \$ MACRO TEST.MAR/NAMES=DEFINITIONS:UPPER, TEST2.MAR/NAMES=DEFINITIONS:LOWER

In this example, global symbol definitions from TEST.MAR are produced in the resulting object file (TEST.OBJ) in uppercase and global symbol definitions from TEST2.MAR are produced in the resulting object file (TEST2.OBJ) in lowercase.

System Dump Analyzer

This chapter describes new System Dump Analyzer (SDA) features.

20.1 TMSCP Symbol

The SDA symbol table now includes the symbol TMSCP. TMSCP (tape mass storage control protocol) represents the address of loadable TMSCP server code, as shown in the following example:

```
SDA> SHOW SYMBOL TMSCP  
TMSCP = 80A35D60 : 000036F0
```

For general information about SDA symbols, see the *VMS System Dump Analyzer Utility Manual*.

20.2 Support for Transaction Processing

The System Dump Analyzer Utility has been modified to provide support for transaction processing. This support is provided by two new SDA commands and by two new qualifiers for the SHOW PROCESS command.

The following SDA commands have been added:

- SHOW LOGS—Displays information about transaction log files currently open for the node.
- SHOW TRANSACTIONS—Displays information about all transactions on the node or about a specific transaction.

In addition, the /PARTICIPANTS and /TRANSACTIONS qualifiers have been added to the SHOW PROCESS command.

The rest of this section describes these SDA commands in more detail.

SHOW LOGS

SHOW LOGS

Displays information about transaction log files currently open for the node.

Format

SHOW LOGS[/qualifier[,...]]

Qualifiers

/DISPLAY=(item [,...])

Specifies the type of information to be displayed. The argument to /DISPLAY can be either a single item or a list. You can specify the following items:

Item	Description
ALL	All transaction log control structure information. This is the default behavior.
OPENS	Transaction log open requests.
READS	Transaction log read requests.
WRITES	Transaction log write requests.

Example

SDA> SHOW LOGS/DISPLAY=(OPENS, WRITES)

The SHOW LOGS command displays the log open request and log write request information for all open transaction logs for the node.

SHOW PROCESS/PARTICIPANTS

Displays information about all transactions for the process.

Format

SHOW PROCESS/PARTICIPANTS[=DISPLAY=(item [...])]

Description

The /PARTICIPANTS qualifier specifies the type of information to be displayed. The argument to DISPLAY can be either a single item or a list. You can specify the following items:

Item	Description
ALL	All transaction control structures for the transactions. This is the default behavior.
BRANCHES	Control structures for branches of the transactions.
PARTICIPANTS	Control structures for resource managers participating in the transactions.
THREADS	Control structures for threads of the transactions.
TRANSACTIONS	Transaction control structures for the transactions.

Example

SDA> SHOW PROCESS/PARTICIPANTS=DISPLAY=PARTICIPANTS

The SHOW PROCESS command displays the control structures for resource managers participating in all transactions in the current process.

SHOW PROCESS/TRANSACTIONS

SHOW PROCESS/TRANSACTIONS

Displays information about all transactions, or a specific transaction, for the process.

Format

SHOW PROCESS/TRANSACTIONS=(option [...])

Description

The /TRANSACTIONS qualifier displays information about all transactions, or a specific transaction, for the process. You can specify the following two options either together or separately.

- **DISPLAY=(item [...])**

Specifies the type of information to be displayed. The argument to **DISPLAY** can be either a single item or a list. You can specify the following items:

Item	Description
ALL	All transaction control structures for the specified transaction. This is the default behavior.
BRANCHES	Control structures for branches of the specified transaction.
PARTICIPANTS	Control structures for resource managers participating in the specified transaction.
THREADS	Control structures for threads of the specified transaction.
TRANSACTIONS	Transaction control structures for the specified transaction.

- **TID=tid**

Specifies the transaction for which information is to be displayed. If you omit the **TID** option, the **SHOW PROCESS/TRANSACTIONS** command displays information about all transactions for the process.

If you omit these options, the **SHOW PROCESS/TRANSACTIONS** command displays all information about all transactions for the process.

Note that the **SHOW PROCESS/TRANSACTIONS** and **SHOW PROCESS /PARTICIPANTS** commands are similar. They display the same information about transactions, but the **SHOW PROCESS/TRANSACTIONS** command displays information about a transaction queue and the **SHOW PROCESS /PARTICIPANTS** command displays information about a resource manager queue.

Examples

SDA> **SHOW PROCESS/TRANSACTIONS=TID=FAC21DE2-BA88-0092-8FA6-00000000B24B**

The **SHOW PROCESS** command displays all transaction control structures for the specified transaction in the current process.

SDA> **SHOW PROCESS/TRANSACTIONS=(DISPLAY=PARTICIPANTS,TID=FAC21DE2-BA88-0092-8FA6-B24B)**

The **SHOW PROCESS** command displays the control structures for resource managers participating in the specified transaction in the current process.

SHOW TRANSACTIONS

SHOW TRANSACTIONS

Displays information about all transactions on the node or about a specific transaction.

Format

SHOW TRANSACTIONS[/qualifier[,...]]

Qualifiers

/DISPLAY=(item [,...])

Specifies the type of information to be displayed. The argument to /DISPLAY can be either a single item or a list. You can specify the following items:

Item	Description
ALL	All transaction control structures for the specified transaction. This is the default behavior.
BRANCHES	Control structures for branches of the specified transaction.
PARTICIPANTS	Control structures for resource managers participating in the specified transaction.
THREADS	Control structures for threads of the specified transaction.
TRANSACTIONS	Transaction control structures for the specified transaction.

/SUMMARY

Displays statistics for transactions on the node. The /SUMMARY qualifier cannot be used with the /TID or /DISPLAY qualifiers.

/TID=tid

Specifies the transaction for which information is to be displayed. If you omit the /TID qualifier, the SHOW TRANSACTIONS command displays information about all transactions on the node.

Examples

```
SDA> SHOW TRANSACTIONS/TID=FAC21DE2-BA88-0092-8FA6-00000000B24B
```

The SHOW TRANSACTIONS command displays all the transaction control structure information for the transaction identified by the transaction identifier (TID).

```
SDA> SHOW TRANSACTIONS/DISPLAY=(PARTICIPANTS, BRANCHES)
```

The SHOW TRANSACTIONS command displays the transaction branch and resource manager information for all transactions on the node.

Mailbox Driver

This chapter describes new features of the mailbox driver. A mailbox driver is a set of routines that VMS uses to facilitate communication among processes.

21.1 Unidirectional Mailboxes

Prior to this release of VMS, channels assigned to mailboxes have always been bidirectional (read/write) channels. This means that both read requests and write requests can be issued to the channel. Channels can now be assigned to mailboxes as unidirectional, either read only or write only. This allows for greater synchronization between users of the mailbox.

To specify a unidirectional channel to the mailbox, **flags** parameters have been added to the \$CREMBX and \$ASSIGN system services. If the **flags** parameter is not specified, or is zero, then the channel assigned to the mailbox is bidirectional (read/write).

See the *VMS System Services Reference Manual* for a syntax description of the \$CREMBX and \$ASSIGN system services.

21.2 Mailbox Driver Functions and Modifiers

The following sections describe the new mailbox driver *wait for writer and reader* functions and the new IO\$M_WRITERCHECK, IO\$M_READERCHECK, and IO\$M_STREAM function modifiers.

21.2.1 Wait for Writer/Reader Function

The wait for writer or wait for reader mailbox driver functions wait until a channel is assigned to the mailbox with the requested access direction. The function returns immediately if a channel is already assigned to the mailbox with the proper access direction. The function always returns immediately if issued on a bidirectional (read/write) mailbox channel (any channel assigned bidirectionally to the mailbox satisfies both wait for writer and wait for reader requests).

The wait for writer/reader functions require the same synchronization techniques as all other \$QIO and \$QIOW functions. The wait for writer/reader functions behave identically with either the \$QIO or \$QIOW function.

The following function codes and modifiers are provided:

- IO\$_SETMODE!IO\$M_READERWAIT—Wait for a read channel to be assigned to the mailbox.
- IO\$_SETMODE!IO\$M_WRITERWAIT—Wait for a write channel to be assigned to the mailbox.

These function codes have no arguments. Once they are enabled and the \$QIO operation has completed, they must be explicitly reenabled.

21.2.2 IO\$M_WRITERCHECK Function Modifier

The IO\$M_WRITERCHECK function modifier for the read mailbox function completes the I/O operation, with SS\$_NOWRITER status, if the mailbox is empty and no write channels are assigned to the mailbox. If no writer is assigned to the mailbox when the \$QIO is issued and no data is in the mailbox, the \$QIO completes immediately. If no data is in the mailbox, but there is a writer assigned, the \$QIO operation completes when either a message is written or all writers deassign their channel to the mailbox. IO\$M_WRITERCHECK is a meaningless function if the channel on which it is issued is read/write because there is always a writer assigned.

21.2.3 IO\$M_READERCHECK Function Modifier

The IO\$M_READERCHECK function modifier for the write and write end-of-file message mailbox function completes the I/O operation immediately, with SS\$_NOREADER status, if no read channels are assigned to the mailbox. If a \$QIO write request with IO\$M_READERCHECK is issued and is outstanding, and all read channels assigned to the mailbox are then deassigned, the \$QIO completes with SS\$_NOREADER status. IO\$M_READERCHECK is meaningless if the channel on which it is issued is read/write because there is always a reader assigned. If SS\$_NOREADER is returned for a write request, the \$QIO write operation does not place any data in the mailbox. If SS\$_NOREADER is returned for a write end-of-file message request, the \$QIO write operation does not place the end-of-file marker in the mailbox.

21.2.4 IO\$M_STREAM Function Modifier

The IO\$M_STREAM function modifier for the read mailbox function ignores QIO record boundaries. The read operation transfers message data to the user's buffer until either the number of bytes specified by the P2 argument are transferred (P2 represents the maximum allowed buffer size, in bytes), all message data currently in the mailbox is transferred, or an end-of-file message is encountered. If a WRITEOF message is within the records required to be read in order to fulfill the request for P2 bytes, the read request terminates successfully with the bytes it was able to read before finding the WRITEOF message, and the end-of-file message becomes the next message to be read. The next read request for greater than zero bytes processes the end-of-file message. \$QIO read stream can return fewer than P2 bytes with a return value of SS\$_NORMAL if the mailbox is emptied by the \$QIO read stream request or a WRITEOF message is encountered.

A READ IO\$M_STREAM request (without IO\$M_NOW specified) on an empty mailbox waits until some data has been written to the mailbox. It terminates with

- Zero bytes read if the next data written is an end-of-file message.
- Fewer than P2 bytes read if the next data written is less than P2 bytes but greater than zero bytes. (READ IO\$M_STREAM ignores write requests of zero bytes.)
- P2 bytes read if the next data written is greater than or equal to P2 bytes.

If a \$QIO read stream request is fulfilled by multiple \$QIO write requests, the sender PID returned in the IOSB of \$QIO read stream reflects the first write request. \$QIO read stream is then charged the buffer quota for the request. This buffer quota is released when the read request is met. A \$QIO read stream

request that would cause the buffer quota to be exceeded for the mailbox when the mailbox has no write requests pending returns an `SS$_EXQUOTA` error.

A `$QIO` read stream request that would cause the buffer quota to be exceeded still executes if the buffer quota is occupied by write requests. This is because allowing the mailbox to temporarily exceed the buffer quota frees the buffer quota. Similarly, a `$QIO` write request that would cause the buffer quota to be exceeded still executes if the buffer quota is occupied by read stream requests.

Read requests of zero bytes are handled differently depending on which functional modifiers are specified. If `IO$_M_STREAM` is specified, then the `$QIO` returns `SS$_NORMAL` with zero bytes read. The contents of the mailbox remain exactly as they were before the `$QIO` was issued. A `$QIO` read stream request of zero bytes does not remove a zero-byte record, nor does it remove an end-of-file marker. However, if `IO$_M_STREAM` is not specified, then `$QIO` can return either `SS$_NORMAL` (if zero bytes were written with the corresponding `$QIO` write request), `SS$_BUFFEROVF` (if the corresponding `$QIO` write request wrote more than zero bytes), or `SS$_ENDOFFILE` (if a `WRITEOF` function was performed as the corresponding `$QIO` write function). For a zero-byte, nonstreaming read request, a record is actually removed from the mailbox in order to meet the `$QIO` read request. Note that even though a record is removed, the corresponding `$QIO` write request should still be performed.

\$QIO Support for Moving Disk Files

VMS Version 5.5 provides a new ACP-QIO subfunction called movefile that permits you to move the contents of a file, or part of the contents of a file, to a new disk location. This subfunction could, for example, form the basis of a disk defragmentation application.

You can disable movefile operations on specific user files by specifying the /NOMOVE qualifier on the SET FILE command. The DIRECTORY/FULL and the DUMP/HEADER commands have been modified to inform you if movefile operations are disabled on a file. See Chapter 11 for more information about the DCL commands that have been enhanced to support movefile operations.

22.1 Calling the Movefile Subfunction

A program can invoke a movefile subfunction by issuing a QIO request using the function code IO\$_MODIFY and the function modifier IO\$_MOVEFILE. This section describes the input parameters that control the processing of movefile operations and also how the movefile subfunction works.

22.1.1 Input Parameters

Table 22-1 lists the FIB (file identification block) fields that control the processing of a movefile subfunction. (See the *VMS I/O User's Reference Manual: Part I* for a description of the FIB.)

Table 22-1 FIB Fields (Movefile)

Field	Field Values	Meaning
FIB\$L_ACCTL	FIB\$V_NOVERIFY	This movefile flag inhibits comparison of the moved blocks. If this flag is clear, the movefile operation verifies that the operation was carried out correctly by comparing the moved blocks to the original blocks.
FIB\$W_FID		Specifies the file identification of the file to be moved.
FIB\$W_EXCTL		Movefile control flags. The following flags apply to the movefile operation. All other FIB\$W_EXCTL flags must be clear.
	FIB\$M_ALCON	Specifies that the movefile operation must allocate contiguous disk space to the moved blocks. If the necessary contiguous space is not available, the movefile operation fails.
		The movefile operation sets this flag if the file was previously marked contiguous.

(continued on next page)

\$QIO Support for Moving Disk Files

22.1 Calling the Movefile Subfunction

Table 22-1 (Cont.) FIB Fields (Movefile)

Field	Field Values	Meaning
	FIB\$M_ALCONB	Specifies that the movefile operation should do its best to allocate contiguous disk space to the moved blocks. That is, if the movefile operation cannot allocate contiguous space to all the moved blocks, it allocates contiguous space to as many of the blocks as possible. The movefile operation sets this flag if the file was previously marked contiguous best try.
	FIB\$M_FILCON	Specifies that the entire file must be made contiguous. Do not set this flag without also setting the FIB\$M_ALCON flag. If the FIB\$M_FILCON flag is set, and either the FIB\$M_ALCON flag is clear or the file would not be made contiguous by moving the specified virtual blocks, the movefile operation fails. The movefile operation sets this flag if the file was previously marked contiguous.
	FIB\$V_NOPLACE	Specifies that placement information will not be recorded in the file header. If this flag is clear, and you specify exact placement for the moved blocks, placement information for those blocks will be recorded in the file header. If this flag is set, the placement information will not be recorded. You specify exact placement through the FIB\$M_EXACT, FIB\$C_LBN, and FIB\$L_LOC_ADDR fields.
FIB\$B_ALOPTS		Flags that control the placement of the allocated blocks. Currently, only the FIB\$M_EXACT flag applies to the movefile operation. All other FIB\$B_ALOPTS flags must be clear.
	FIB\$M_EXACT	Set to require exact placement. If this flag is set and the specified blocks are not available, the movefile operation fails.
FIB\$B_ALALIGN		Contains the interpretation mode of the allocation field (FIB\$W_ALLOC). You can specify a field value of zero or you can specify the symbolic value FIB\$C_LBN. If you specify zero, the allocation field is ignored.
	FIB\$C_LBN	If the FIB\$M_EXACT flag is also set, indicates that the FIB\$L_LOC_ADDR subfield contains the starting logical address to which the blocks are moved.
FIB\$W_ALLOC		Contains the desired location of the blocks being allocated. Interpretation of the field is controlled by the FIB\$B_ALALIGN field.
	FIB\$B_LOC_RVN	Placement relative volume number (RVN).
	FIB\$L_LOC_ADDR	If the FIB\$C_LBN and FIB\$M_EXACT flags are set, specifies the starting logical address to which the blocks are moved.

(continued on next page)

Table 22-1 (Cont.) FIB Fields (Movefile)

Field	Field Values	Meaning
FIB\$L_MOV_SVBN		Specifies the virtual block number (VBN) of the first block to be moved. The starting virtual block number must correspond to the first block of a disk cluster. The value must be greater than zero and it must not exceed the number of virtual blocks allocated to the file. If you specify an invalid value, the movefile operation fails.
FIB\$L_MOV_VBNCNT		Specifies the number of consecutive virtual blocks to be moved. This value must be a multiple of the disk cluster size and it must not exceed the difference between the greatest VBN (virtual block number) allocated to the file and the FIB\$L_MOV_SVBN value. If you specify a value of zero, the movefile operation moves all the virtual blocks between the FIB\$L_MOV_SVBN value and the greatest VBN. If you specify an invalid value, the movefile operation fails.

22.1.2 Operation

A program can perform a movefile operation on a file if the following conditions are met:

- The program has write and control access to the file.
- The file is closed.
- Movefile operations are not disabled on the file.
Movefile operations are automatically disabled on critical system files. You can disable movefile operations on specific user files by specifying the /NOMOVE qualifier on the SET FILE command. See Chapter 11.
- The operation is not interrupted.
If the movefile operation is interrupted by any other operation, the movefile operation aborts and the file remains in its original position.
- The source and target locations are on the same disk.
You cannot transfer blocks from one volume to another and you cannot move blocks spanning more than one volume.

The movefile operation moves a specified number of consecutive virtual blocks to new logical blocks on the disk, beginning with the virtual block specified in the FIB\$L_SVBN field.

The number of blocks moved is specified in the FIB\$L_VBNCNT field. To move an entire file, specify FIB\$L_VBNCNT as 0 and FIB\$L_SVBN as 1.

To specify a starting logical block number for the moved blocks, write the logical block address in the FIB\$L_LOC_ADDR subfield and set the FIB\$C_LBN and the FIB\$M_EXACT flags.

\$QIO Support for Moving Disk Files

22.1 Calling the Movefile Subfunction

If the file was previously marked contiguous, the movefile operation sets the FIB\$M_ALCON and FIB\$M_FILCON flags. This ensures that a contiguous file is not fragmented by a movefile operation. Similarly, if the file was previously marked contiguous best try, the movefile operation sets the FIB\$M_ALCONB flag.

For virtual blocks beyond the file's highwater mark, the movefile operation allocates new logical blocks but does not copy the contents. The position of the file's highwater mark remains unchanged.

VMS Version 5.4-3 Features

This appendix describes features introduced with VMS Version 5.4-3 but not yet documented in other printed manuals.

A.1 Summary of New VMS Version 5.4-3 Software Features

Table A-1 provides a summary of the VMS Version 5.4-3 software features. For information about new and enhanced hardware, see the *VMS Version 5.4-3 Release Notes*.

Table A-1 Summary of VMS Version 5.4-3 Software Features

VMS Version 5.4-3 Systemwide Features	
Backup Utility	With this version of VMS, you can make the backup tape drive available for other operations before the backup procedure completes and you can now mount a tape that is protected by a volume-accessibility character or a tape created by HSC backup. VMS Version 5.4-3 also gives you several label-processing options for non-ANSI tapes and improved error reporting from disk and tape drivers.
Fiber Distributed Data Interface (FDDI)	Changes to the VMS programming interface for local area networks (LANs) provide support for FDDI, Digital's next generation of local area networking. FDDI can accommodate a data rate 10 times that of Ethernet and has a significantly larger LAN diameter, frame size, and message size than Ethernet. Only minimal changes need to be made to existing Ethernet applications for them to run on FDDI. The Network Control Program (NCP) supports FDDI by providing a set of counters for monitoring FDDI line errors and line performance.
VMSINSTAL	The RUN-IMAGE callback now lets you defer running the image.
Proactive Memory Reclamation from Idle Processes	VMS now supports proactive memory reclamation, a memory management policy that allows the operating system to reclaim memory from long-waiting processes and periodically waking processes. Memory is reclaimed proactively from an inactive process when a deficit is first detected but before the memory resource is depleted. Prior versions of the VMS operating system trimmed processes with first- and second-level working-set trimming before resorting to swapping.

(continued on next page)

VMS Version 5.4-3 Features

A.1 Summary of New VMS Version 5.4-3 Software Features

Table A-1 (Cont.) Summary of VMS Version 5.4-3 Software Features

VMS Version 5.4-3 Systemwide Features	
Open-Bus Driver Support	<p>VMS provides new open-bus driver support features for VMEbus and SCSI bus device driver programming needs. The support for VMEbus device connections to various XMI-based VAX processors permits the writing of third-party VMEbus device drivers.</p> <p>For SCSI device drivers, the support specifically includes programming for the NCR 53C94 Controller.</p>
VMS Local Area VAXcluster Software	<p>VMS provides support for up to four local area network (LAN) adapters on each local area VAXcluster node, including Ethernet adapters and FDDI adapters.</p> <p>New sample programs and related subroutine packages are provided in SYS\$EXAMPLES to start and stop the local area VAXcluster protocol on a LAN adapter and to enable local area VAXcluster network failure analysis. See the <i>VMS VAXcluster Manual</i> for information about these features.</p>
VAX Ada RTL	<p>Additional precision has been provided for delay statements and an additional delete capability has been provided for Ada I/O packages.</p>
System Dump Analyzer	<p>Changes to the SDA command SHOW PORTS now allow you to view the data structures that the multiadapter local area cluster uses.</p>

A.2 VMS Version 5.4-3 System Management Features

This section contains information about new features for the VMS Version 5.4-3 operating system that is of interest to system managers.

A.2.1 Backup Utility

This section describes new features and options for the VMS Backup Utility (BACKUP) for VMS Version 5.4-3.

A.2.1.1 /RELEASE_TAPE Qualifier

The /RELEASE_TAPE qualifier is new for the DCL command BACKUP. /RELEASE_TAPE dismounts and unloads a tape after a backup save operation writes a save set to the tape.

By using the /RELEASE_TAPE qualifier with either the /DELETE or /RECORD qualifier, you can make the tape drive available for other operations before the backup procedure completes. For example, you could use the following command to back up a disk:

```
$ BACKUP/IMAGE/RECORD/RELEASE_TAPE DUA1: MUA0:BACK.BCK
```

By using the /RELEASE_TAPE and /RECORD qualifiers, the Backup Utility dismounts and unloads the tape before it performs the action of the /RECORD qualifier.

In the following example, the /RELEASE_TAPE qualifier dismounts and unloads the tape before the /DELETE qualifier performs its action:


```
$ ALLOCATE MUA0: TAPE
$ BACKUP/DELETE/RELEASE_TAPE/LOG DUA1:[MAIN...] MUA0:MAIN.BCK
.
.
$ DEALLOCATE TAPE
```

The tape drive remains allocated until you enter the DEALLOCATE command.

Note that you cannot use the /RECORD and /DELETE qualifiers in the same BACKUP command.

A.2.1.2 ACCESSIBILITY Keyword

The BACKUP command qualifier /IGNORE now accepts a new keyword, ACCESSIBILITY. This keyword allows the Backup Utility to mount a tape that is protected by a volume-accessibility character or a tape created by hierarchical storage controller (HSC) backup. The keyword applies only to tapes. It affects the first tape mounted and all subsequent tapes in the save set.

The following example shows how to use the ACCESSIBILITY keyword:

```
$ INITIALIZE/LABEL=VOLUME_ACCESSIBILITY:"K" MUA1: 29JUN
$ BACKUP/IGNORE=(ACCESSIBILITY) DUA0:[BOOKS...] MUA1:BACKUP.SAV/LABEL=29JUN
```

In this example, the tape is initialized with an accessibility character (K) and a volume label (BACKUP). The BACKUP command mounts the tape, regardless of the accessibility, and performs the backup operation. For more information about tape protection, refer to the *Guide to VMS Files and Devices*.

A.2.1.3 Backup Label Processing Options

In previous VMS versions, the VMS Backup Utility automatically overwrote a non-ANSI-labeled tape during a backup save operation.

With VMS Version 5.4-3, the Backup Utility now provides you with several options when it encounters a tape that has an ANSI label:

```
%MOUNT-I-MOUNTED, DKA0 mounted on _SODAK$MUA0:
%BACKUP-W-MOUNTERR, volume 1 on _SODAK$MUA0 was not mounted because
its label does not match the one requested
Specify option (QUIT, NEW tape or OVERWRITE tape)
BACKUP>
```

Depending on the option you specify, you can quit the backup (QUIT), dismount the old tape and mount a new one (NEW), or overwrite the data on the tape (OVERWRITE).

If you use scratch tapes, which you intend to overwrite, use the /IGNORE=LABEL_PROCESSING qualifier. This suppresses the previous Backup Utility message, which normally occurs if the Backup Utility encounters a tape that does have an ANSI label.

A.2.2 Disk and Tape Class Drivers—Enhanced Error Reporting

In concert with Digital fault management strategy, the disk and tape class drivers have been modified to analyze error messages and to determine whether or not to make an error log entry and to increment the device error count.

The device-specific error counts now accurately reflect the number of errors and are not indicative of the number of error-related messages received.

A.2.3 New NCP Line Counters for FDDI Communications

The fiber data distributed interface (FDDI) is Digital's next generation of local area networking to follow Ethernet. The first VMS device or network adapter for FDDI is the DEC FDDIcontroller 400, called DEMFA, for VAX systems based on XMI (6000/9000 class). The DEMFA and FDDI are supported by DECnet-VAX Phase IV and DECnet-VAX Extensions.

VMS Version 5.4-3 uses new NCP line counters for FDDI communications. You can use these counters to display error and performance statistics about your FDDI line. The following command shows how to display information about an FDDI line:

```
NCP> SHOW LINE MFA-n COUNTERS
```

where $n=0,1,2,\dots$

For more information about FDDI, see *A Primer to FDDI: Fiber Distributed Data Interface* and *Fiber Distributed Data Interface System Level Description*.

The new NCP line counters are described as follows:

Connections completed

Indicates the number of times the PHY Port entered the In Use state, after having completed the initialization process.

Directed beacons received

Indicates the number of times the link received a unique directed beacon. A unique directed beacon is the assertion of Other_Beacon and receipt of particular beacon data.

Duplicate address test failures

Indicates the number of times the duplicate address test failed. That is, how many times it detected that the link address was a duplicate.

Duplicate tokens detected

Indicates the number of times the media access control (MAC) detected a duplicate token either by means of the duplicate token-detection algorithm or by the receipt of a token while already holding one.

Elasticity buffer errors

Indicates the number of times the Elasticity Buffer function in the PHY Port had an overflow or underflow. This indicates a transmit clock error somewhere in the network.

FCI strip errors

Indicates the number of times the receipt of a token terminated a Frame Content Independent Strip.

LCT rejects

Indicates the number of times a connection on this PHY Port was rejected because the Link Confidence Test (LCT) at either end of the physical connection failed. The LCT rejects counter only counts rejections that cause the link to enter into the Watch State. The counter, therefore, indicates the number of *distinct* link quality problems rather than the total length of time such problems persisted.

LEM rejects

Indicates the number of times an active connection on this PHY Port was disconnected because the Link Error Monitor (LEM) at this end of the physical connection rejected the connection or because the Noise timer (TNE) expired.

Link errors

Indicates the total number of "raw" Link Error input events detected by the Link Error Monitor (LEM).

MAC error count

Indicates the total number of times the media access control (MAC) changed the error indicator in a frame from reset to set. This tells you the number of times the local FDDI adapter detected an error in a frame.

MAC frame count

Indicates the total number of frames on the FDDI media, other than tokens.

MAC lost count

Indicates the total number of times a frame (other than a token) was improperly terminated.

Ring beacons initiated

Indicates the number of times this station initiated the ring beacon process.

Ring initializations initiated

Indicates the number of times this station initiated a ring reinitialization.

Ring initializations received

Indicates the number of times another station initiated ring reinitialization.

Ring purge errors

Indicates the number of times the ring purger received a token while still in the ring purge state.

Traces initiated

Indicates the number of times this link initiated the PC-trace process.

Traces received

Indicates the number of times another link initiated the PC-trace process.

A.2.4 FDDI/Ethernet Startup Error Code

In VMS Version 5.4-3, a new error code, SS\$_IVADDR, can be returned from a SETMODE!STARTUP QIO request to the FDDI/Ethernet drivers. The driver returns the code when the requested Ethernet physical address already exists on the extended LAN to which your device is attached.

The following error message is associated with the error code:

%SYSTEM-F-IVADDR, invalid media address

A.2.5 Proactive Reclamation of Memory from Idle Processes

VMS Version 5.4-3 introduces a memory management policy that is designed to reclaim memory proactively from inactive processes when a deficit is first detected but before the memory resource is depleted. The policy allows the operating system to reclaim memory from the following types of idle processes: (1) long-waiting processes and (2) periodically waking processes. Proactive reclamation of memory typically maintains a sufficiently large cache of free pages so that active, demanding processes do not have to wait for reclamation to take place. Therefore, perceived response times are noticeably improved in memory-constrained environments.

In previous versions of VMS, while inactive processes continued to hoard large amounts of memory, active processes sometimes were not allowed to grow when memory was constrained. Very little free memory was available on these systems, so when performing memory-intensive activities, users typically experienced perceptible delays while the system attempted to reclaim memory by trimming and swapping.

Past versions of VMS attempted to trim processes with first- and second-level working-set trimming before resorting to swapping. The conventional wisdom was that swapping resulted in sluggish system performance and poor user response times. By the time the system worked its way to swapping out processes, performance was significantly degraded. All processes, regardless of their activity levels, had been trimmed to either their working-set quota or to SWPOUTPGCNT and the free page list typically hovered near FREELIM. If a process needed to be swapped in, the free page list was frequently too small to satisfy the demand, potentially triggering more swapping. This behavior continued until users became frustrated and logged out of the system. And yet, even with this sluggish behavior, certain inactive processes might still be hoarding relatively large amounts of memory for long periods of time. Clearly, these inactive processes are prime candidates for memory reclamation before memory is exhausted. You can expect overall system performance to improve as the system makes this memory available to active processes.

A.2.5.1 How Is This Policy Enabled?

VMS Version 5.4-3 enables proactive memory reclamation by default. However, using the system parameter MMG_CTLFLAGS, you can enable or disable proactive memory reclamation from periodically waking processes or long-waiting processes or both. The system parameter MMG_CTLFLAGS is bit encoded. Bit <0> enables memory reclamation by trimming periodically waking processes. Bit <1> enables memory reclamation by swapping out long-waiting processes. Therefore, choose a value for MMG_CTLFLAGS from 0 to 3 that sets or clears the low-order bits <0> and <1> to enable or disable, respectively, the policy for either periodically waking processes or long-waiting processes or both.

A.2.5.2 Reclaiming Memory from Long-Waiting Processes

In this instance, the proactive memory reclamation policy involves the swapping out of long-waiting processes when the size of the free page list drops below the value of FREEGOAL. An example of a candidate for this memory management policy is a process that has been in the LEF or HIB state for longer than the number of seconds in the system parameter LONGWAIT.

In VMS Version 5.4-3, with this default policy, when you use such commands as SHOW SYSTEM in memory-constrained environments, the resulting display most likely shows more processes swapped out than it did in previous versions of VMS. This is the expected and desired behavior for this release. (See Section A.2.5.1 for information about enabling the memory reclamation mechanisms.) Swapping out long-waiting processes is triggered when the free list is at or below the value of the system parameter FREEGOAL.

Note

When this policy is active, AUTOGEN sets the system parameter FREEGOAL to a value considerably greater than in previous VMS releases.

By setting FREEGOAL to a larger size, memory reclamation from idle processes is proactively triggered before a memory deficit becomes crucial and thus results in a larger pool of free pages available to active processes. When a process that has been swapped out in this way must be swapped in, it can frequently satisfy its need for pages from the large free page list. For all but the largest consumers of memory, swapping in does not result in perceptible delays.

This mechanism of swapping out long-waiting processes includes a significant change. When shrinking the working set to the value of the SWPOUTPGCNT system parameter, the memory management policy removes pages from the working set but leaves the working-set size (the limit to which pages may be added to the working set) at its current value, rather than reducing it to the value of SWPOUTPGCNT. In this way, when a process is swapped in, it can readily fault the pages it needs without having to rejustify its size through successive adjustments to the working set. This change contributes significantly to the lack of perceptible delays when the process is swapped in.

A.2.5.3 Reclaiming Memory from Periodically Waking Processes

The proactive memory reclamation policy also targets processes that wake periodically, do minimal work, and then return to a sleep state. An example of such a process is a watchdog process. Because it has a periodically waking behavior, a watchdog process is not a candidate for being swapped out but may be a good candidate for memory reclamation. For this kind of process, VMS Version 5.4-3 memory management tracks the relative wait-to-execution time. When the size of the free page list drops below twice the value of FREEGOAL, the system initiates memory reclamation (trimming) of processes that wake periodically. Waiting until the size of the free page list drops below twice the value of FREEGOAL gives this memory reclamation mechanism an opportunity to trim from periodically waking processes before the more aggressive form of swapping begins. If a periodically waking process is idle over 99% of the time and has accumulated 30 seconds of idle time, the proactive memory reclamation policy trims a percentage of the pages in the process's working set as the process reenters a wait state. The working-set size remains unchanged.

A.2.5.3.1 Setting the FREEGOAL Parameter The system parameter FREEGOAL plays the central role in controlling how much memory is reclaimed from idle processes. Setting FREEGOAL to a larger value reclaims more memory; setting FREEGOAL to a smaller value reclaims less. VMS Version 5.4-3 makes FREEGOAL a dynamic parameter so that it can be adjusted in the active parameter set without rebooting.

For information about setting SYSGEN parameters, refer to the *VMS System Generation Utility Manual*. For a discussion of AUTOGEN, refer to the *Guide to Setting Up a VMS System*.

A.2.5.3.2 Sizing Page and Swap Files Because it reclaims memory from idle processes by trimming and swapping, the new memory reclamation policy can potentially increase page and swap file use. On systems running VMS Version 5.4-3, you should make sure your page and swap files are appropriately sized for the potential increase. Refer to the *Guide to Setting Up a VMS System* for information about sizing page and swap files.

VMS Version 5.4-3 Features

A.2 VMS Version 5.4-3 System Management Features

A.2.6 Tape Support

With this release of VMS, support is provided for the following tape devices:

- TA91 cartridge tape device with loader
- TF85 cartridge tape device
- TF857 cartridge tape device with loader
- TF837 cartridge tape device with loader

A.2.7 VMSINSTAL Callback RUN_IMAGE: New Parameter

The VMSINSTAL callback RUN_IMAGE has a new parameter (P4), the Option parameter. This parameter indicates whether the image is to be run immediately or run deferred. Valid values for this parameter are:

- D—Image is run deferred when in safety mode
- I—Image is run immediately, regardless of mode

The following command line uses the new parameter RUN_IMAGE:

```
$ VMS$CALLBACK RUN_IMAGE NAME.EXE "" D
```

A.3 VMS Version 5.4-3 Programming Features

This section contains information about VMS Version 5.4-3 new features that are of interest to programmers.

A.3.1 Open-Bus Driver Support Features

This section describes enhancements that support the use of open-bus drivers developed by third-party users.

Section A.3.1.1 begins with a description of the VMS device support for VMEbus devices. This is followed by a presentation on VME driver routines in Section A.3.1.2 and a sample VME driver program in Section A.3.1.3.

Section A.3.2 describes SCSI device support for the NCR 53C94 controller.

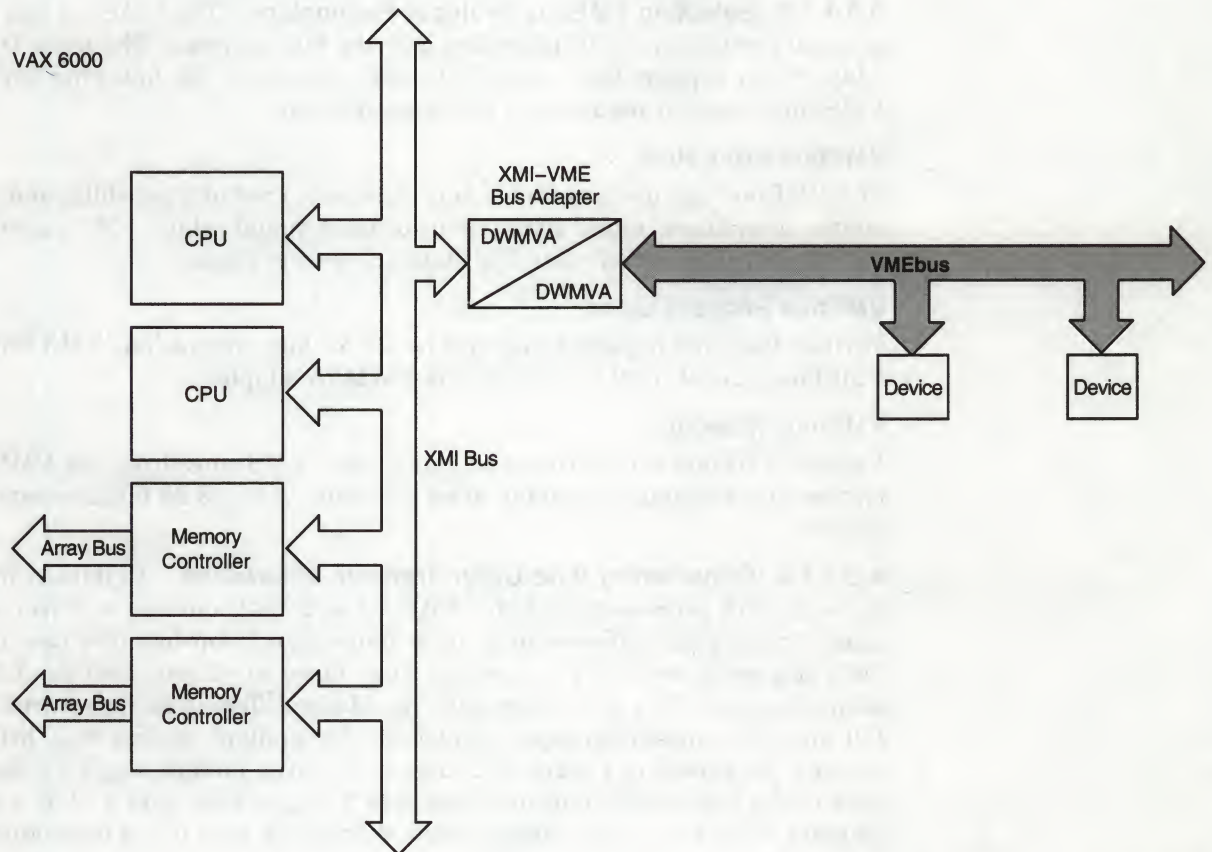
A.3.1.1 VMS Device Support for VMEbus Devices

VMS now supports VMEbus device connections for certain XMI-based VAX processors. The VMS programming support for such connections permits the writing of third-party VMEbus device drivers and provides the required VME device driver I/O routines and optional macro routines.

Two types of I/O operations are supported: direct memory access (DMA) and programmed I/O (PIO) that are VMS resources for VMEbus device data transfers. VMS operating system routines that are specific to VME map VME address space for both DMA and PIO operations and support the setup and delivery of device interrupts. Also included are byte-swapping routines for different hardware needs. The VMS architecture of the VME interface is similar to and conforms to the standards of I/O subsystems such as the UNIBUS and Q-22 bus models described in the *VMS Device Support Manual* and the *VMS Device Support Reference Manual*.

A.3.1.1.1 Hardware Environment The VMEbus device support option is now offered on the VAX 6000 series systems. The VAX 6000 CPU and memory employ an XMI bus to interconnect with I/O adapters. The option incorporates an XMI-to-VMEbus (DWMVA) adapter and a 6U (double-height) VME controller module. The DWMVA adapter supports 32 bits of both address and data buses and conforms to ANSI/IEEE STD 1014. The block diagram in Figure A-1 shows the system based on the XMI/VMEbus with a VAX 6000 CPU and memory.

Figure A-1 System Based on XMI/VMEbus



ZK-3728A-GE

A.3.1.1.2 Associated Documents In addition to the material in this section, you should have an understanding of the information in the following documents:

- *VMS Device Support Manual*, which describes the components of a VMS device driver and the basic rules to which device drivers supplied by vendors other than Digital must adhere.
- *VMS Device Support Reference Manual*, which describes VMS data structures, macros, routines, and driver entry points.
- *DWMVA VME Adapter Technical Manual* (EK-DWMVA-TM-001), which describes the DWMVA adapter and Digital's XMI-to-VMEbus implementation. Information concerning specific driver requirements to implement the hardware/software adapter options is also provided.

VMS Version 5.4-3 Features

A.3 VMS Version 5.4-3 Programming Features

- *An American National Standard—IEEE Standard for a Versatile Backplane Bus: VMEbus* (ANSI/IEEE Std 1014), ISBN 0-471-61601-X

You may need to refer to material in the following manuals for help in certain aspects of application and driver programming:

- *VMS System Services Reference Manual* for a description of the high-level language interface to the I/O subsystem of the VMS operating system
- *VMS System Dump Analyzer Utility Manual* for assistance in investigating system failures

A.3.1.1.3 Selecting VMEbus Protocol Parameters The VMEbus has selectable protocol parameters that determine how the bus operates. Though a DWMVA bus adapter can support the various selectable functions, the following fixed set of VMS initialization parameters has been selected:

VMEbus Arbitration

The VMEbus can operate under four different types of bus arbitration schemes: single, prioritized, round-robin, or prioritized round-robin. VMS currently initializes to the round-robin VMEbus arbitration mode.

VMEbus Request Level

Various VMEbus request levels can be set for bus arbitration. VMS initializes the VMEbus request level to BR3 for the DWMVA adapter.

VMEbus Timeout

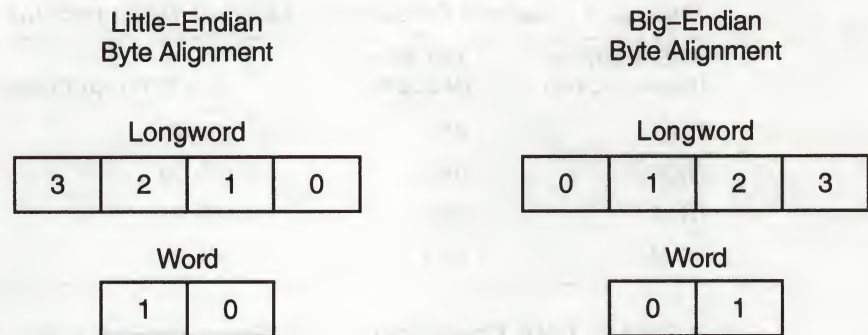
Various VMEbus access timeouts can be set. VMS initializes the VMEbus transaction timeout parameter to be the same value (3.28 milliseconds) for all drivers.

A.3.1.1.4 Considering Byte-Order Transfer Differences In data transfers between VAX processors and the VMEbus and VME devices, a driver writer must consider the different byte order (most-significant-byte first/last or right/left) of a given word or longword between buses of various devices of different manufacturers. The byte-order patterns of the different manufactured devices fall into two opposing groups, defined as: "big-endian" devices and "little-endian" devices. As shown in Figure A-2, byte 3 of a little-endian longword aligns with byte 0 of a big-endian longword and byte 2 aligns with byte 1. For a word transfer, byte 1 of a little-endian word aligns with byte 0 of a big-endian word.

When a VAX VMS driver (which employs a little-endian, byte-aligned bus) performs write transfers of data to a VME device's register or memory location (which is big-endian, byte aligned), the writer must swap the bytes to account for the VME byte lanes. For example, for CSR loading of a VME big-endian device, all data transferred must be byte swapped, performing byte-swap operations on both the write transfers and the read transfers. For the specific DMA and PIO byte-alignment requirements, refer to both the device and adapter technical manuals.

The SWAPWORD and SWAPLONG macros use a register as input and swap the little-endian data to big-endian data so that the big-endian device will receive the correctly ordered data in its register or memory location. In addition, byte-swap routines for words and longwords are provided and described in Section A.3.1.2.

Figure A-2 Little-Endian Versus Big-Endian Byte Alignment



ZK-3729A-GE

A.3.1.1.5 Handling Interrupts VAX peripheral devices request interrupts at IPLs 20 to 23 because device interrupts need to preempt most user and VMS software functions. For the VME subsystem, the VAX 6000 power-up default sets four VME interrupt request levels to four XMI priority levels with read-acknowledge signal mode enabled. In the reinitialization section established by the DPT_STORE macro, the driver prologue table holds the address of one or more interrupt service routines. Each interrupt service routine corresponds to an interrupt vector on an I/O bus. For further information about interrupt service routines, refer to the *VMS Device Support Manual*.

The VAX 6000 employs direct-vector interrupt dispatching (see the *VMS Device Support Manual*). Vector addresses are established during system generation with the CONNECT command (see Section A.3.1.1.10).

When an interrupt occurs, a VME interrupt vector on the bus from a specific VME device is read by the CPU. The CPU then calls the appropriate driver interrupt service routine by using the VME interrupt vector address. The VME architecture permits either single or multiple interrupt handlers on a single VMEbus. The multiple handler is referred to as a distributed handler system.

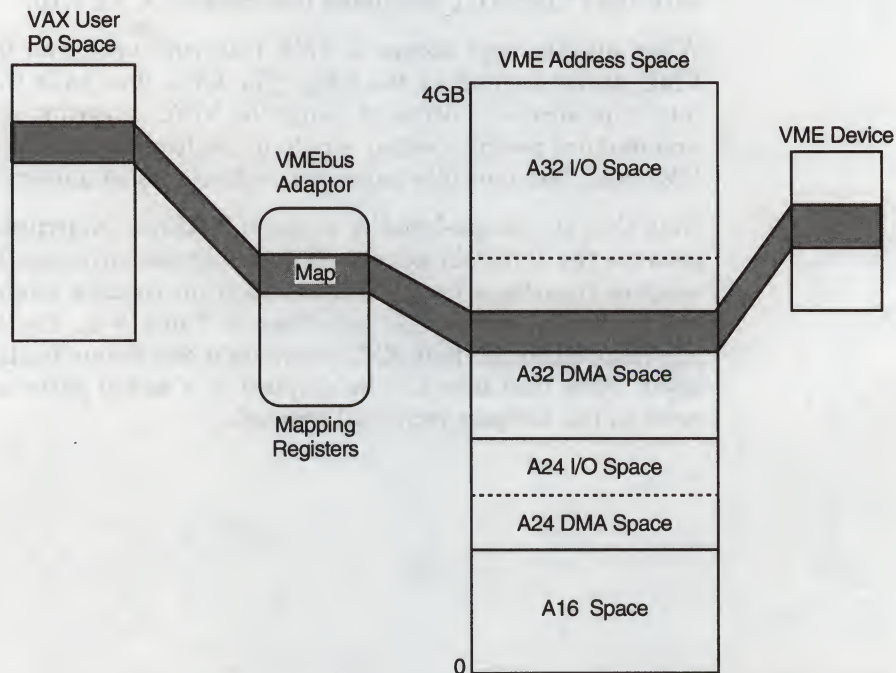
Note that the single-handler mode of VMEbus interrupts is configured by default because the DWMVA adapter is typically the only handler on the VMEbus. The adapter translates four VMEbus interrupt request levels (IRQ7—IRQ4) to XMI bus requests (BR7—BR4), as shown in Table A-2. The four VME request levels are mapped to the four XMI levels on a one-to-one basis at system powerup or reset. Note that they can be mapped in a mixed pattern. For more information, refer to the adapter technical manual.

Table A-2 Mapped Defaults for XMI and VME Interrupt Request Levels

VME Interrupt Request Level	XMI Bus Requests	VAX Interrupt Priority Level
IRQ7	BR7	IPL 23
IRQ6	BR6	IPL 22
IRQ5	BR5	IPL 21
IRQ4	BR4	IPL 20

A.3.1.1.6 DMA Operations The direct memory access (DMA) I/O operation of a VAX host system permits devices and device drivers to exchange large amounts of data. DMA operations for VMEbus devices are similar to the Q-22 bus DMA operations described in the *VMS Device Support Manual*. As shown in Figure A-3, the VMEbus adapter sends DMA data through the direct-DMA path between the VAX host and the VME device. The direct data path (DDP) allows VME transfers to randomly ordered physical addresses. The direct data path maps each VME I/O transfer to a backplane interconnect cycle. The VME address space varies according to the specific VME device and is identified as A16, A24, or A32 space. A32 is the largest, allowing up to 4 gigabytes of space using 32-bit addresses. A24 space is addressable with 24-bit addresses and A16 space is addressable with 16-bit addresses. Note that DMA operation is not permitted with A16 devices.

Figure A-3 VMEbus DMA to and from VAX Host



ZK-3752A-GE

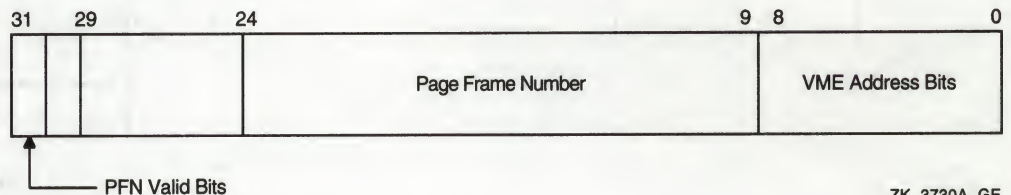
A DMA transaction initiated by the VME device to locations in VAX-XMI memory can consist of 1-, 2-, 3-, or 4-byte single-access transfer cycles or 1-, 2-, or 4-byte read and write cycles in block mode. Up to 256 bytes (per block) of VME data can be transferred to the adapter toward VAX memory. Because VAX-XMI DMA supports quadword, octaword, and hexword data transfers, the adapter buffers the VMEbus blocks into octawords for contiguous locations in VAX memory.

There are three operating system routines provided for VME DMA operations:

- IOC\$ALOVMEMAP_DMA or IOC\$ALOVMEMAP_DMAN
- IOC\$LOADVMEMAP_DMA or IOC\$LOADVMEMAP_DMAN
- IOC\$RELVMEMAP_DMA or IOC\$RELVMEMAP_DMAN

A driver that performs direct DMA transfers to and from VAX memory must allocate a set of map registers (IOC\$ALOVMEMAP_DMA routine). As shown in Figure A-4, a field in each map register identifies the VAX page-frame number corresponding to the VME space address that the map register represents. When a DMA map register is loaded (IOC\$LOADVMEMAP_DMA routine), one VAX page (512 bytes) of VME space is mapped into the VMS address space. Once mapped, VME devices are then free to access this VMS memory with DMA read and write cycles. For more information about the DMA routines, see Section A.3.1.2.

Figure A-4 VMEbus Map Register



ZK-3730A-GE

Note that a DWMVA adapter contains 64K map registers, each of which maps only 512 bytes (one VAX page each). Therefore, only the lower 32MB of VME space can be mapped to VMS address space if VME DMA to VAX is required. However, this does not limit DMA space between VME devices contained on the same VMEbus.

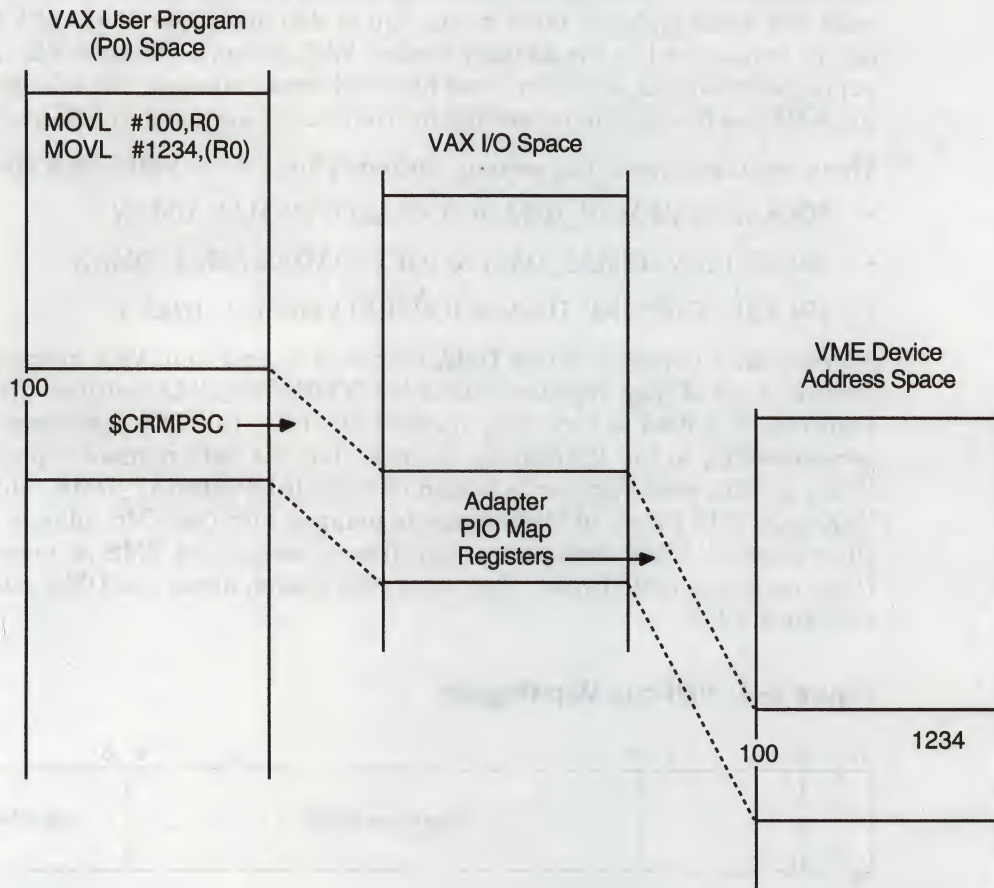
When certain flags are set by the loading routine (IOC\$LOADVMEMAP_DMA or IOC\$LOADVMEMAP_DMAN), the map registers can also specify byte swapping of words or longwords on incoming and outgoing VMS data and/or provide read-modify-write access on a per page basis.

A.3.1.1.7 Programmed I/O Operations and I/O Mapping VMS programs can interface with a VME I/O subsystem by mapping to VAX I/O address space. The VAX CPU accesses the VMEbus address space by loading a set of programmed I/O (PIO) map registers that contain the VMEbus PFN and access information (see the adapter technical manual). The VMS program calls the \$CRMPSC system service to map the PIO map register section in VAX I/O space. The PIO map registers are assigned permanent VAX I/O space locations, so when the CPU reads or writes an I/O space location, it will access the mapped VMEbus address, as shown in Figure A-5.

VMS Version 5.4-3 Features

A.3 VMS Version 5.4-3 Programming Features

Figure A-5 Mapping of Programmed I/O Access from User Space



ZK-3731A-GE

As shown previously in Figure A-3, depending on the device, VME memory space or address ranges vary. There are three modes of PIO access to a VME device's address space from a VMS program:

- Short supervisor access
- Standard supervisor access
- Extended supervisor access

The short supervisor access identifies to a VME address space of 64K bytes that requires 16-bit addresses (A16). Standard supervisor access identifies to a 24-bit address (A24) with space of 16M bytes and Extended supervisor access identifies to a 32-bit address (A32) for access with large space beyond 16M bytes. Refer to the device's specific manual for memory requirements.

One PIO map register is allocated to the system when the system is booted mapping the lower 64K bytes of VME short space into VMS system space. The VME memory access is set up in short space with word-access (A16) mode enabled. Refer to the adapter technical manual for the physical starting address of the I/O adapter space. The CSR offset value (specified when loading the driver) is limited to a word so that the maximum range would be from 0 to 64K. If the CSR for a device is located in the lower 64K and requires word access, the proper VMS system address will be passed to the driver by established Q-bus

driver methods (such as IDB\$L_CSR and R4 of the unit/controller initialization routines). All other CSR accesses must be handled by the driver as a special event.

There are three operating system routines provided for programmed I/O VMEbus support:

- IOC\$ALOVMEMAP_PIO
- IOC\$LOADVMEMAP_PIO
- IOC\$RELVMEMAP_PIO

The routines supplied for PIO map registers are similar to the ones supplied for DMA. These routines simplify the driver coding that allocates, loads, and releases the map registers. For more information, see Section A.3.1.2.

A.3.1.1.8 Coding a VMEbus Device Driver Write the device driver in one or more source files coding to the requirements of the *VMS Device Support Manual*. A sample VME driver in Section A.3.1.3 provides a code example of a DR11 VME driver with a DMA interface. In addition to the DR11 driver, other VME driver samples are provided in SYS\$EXAMPLES. Table A-3 lists standard driver routines that you might need to provide entry points for VMS in your program. The routines are described in more detail in the *VMS Device Support Reference Manual*.

Table A-3 Driver Entry Point Routines

Routine	Description
Alternate Start I/O	Initiates activity on a device that can support multiple, concurrent I/O operations and synchronizes access to its UCB.
Cancel I/O	Prevents further device-specific processing of the I/O request currently being processed on a device.
Controller Initialization	Prepares a controller for operation.
Driver Unloading	Prepares a driver for unloading or reloading.
FDT (\$QIO Handling)	Performs any device-dependent activities needed to prepare the I/O database to process an I/O request.
Interrupt Service	Processes interrupts generated by a device.
Register Dumping	Copies the contents of a device's registers to an error message buffer or a diagnostic buffer.
Start I/O	Activates a device to process a requested I/O function.
Timeout Handling	Takes whatever action is necessary when a device has not yet responded to a request for device activity and the time allowed for a response has expired.
Unit Delivery	For controllers that can control a variable number of device units, determines which specific devices are present and available for inclusion in the system's configuration.
Unit Initialization	Prepares a device for operation and, in the case of a device on a dedicated controller, initializes the controller.

The VME support routines described in Section A.3.1.2 are supplied in a separate object library to which the driver must link. Place the PSECT (program section) containing the VME support routines (\$\$\$112_VME_SUPPORT_ROUTINES) after the prologue PSECT and just ahead of the main driver code. For information about other PSECTs, see the *VMS Device Support Manual*.

VMS Version 5.4-3 Features

A.3 VMS Version 5.4-3 Programming Features

Porting from Drivers Based on the UNIX System

For the task of porting drivers based on the UNIX system to VMS, Table A-4 provides a list of associated notions in a driver translation from UNIX to VMS. Note in some cases, these notions are loosely connected and may not provide a pure relationship.

Table A-4 Driver Notions Porting from UNIX to VMS

UNIX Name	UNIX Description	VMS Name	VMS Description
u	User current process structure	PCB	Process control block
iobuf	Device table (block device control block xxxxtab.xxxx)	UCB	Unit control block
buf	Block I/O descriptor	IRP	I/O request packet
xx_device	Device data structure (CSRs and data registers)	UCB Extension	
clist	Character driver temporary storage (line accumulator)	SILO buffer	Service in logical order buffer for the channel
cblocks	24-byte packets		Data input packets in serial channel
dev_init	Device initialization table in conf.c	DPT	Device prologue table
bdevsw	Device switch tables in conf.c for block driver	DDT	Driver dispatch table
cdevsw	Device switch tables in conf.c for character driver	DDT	Driver dispatch table
dev_addr	Device address table in conf.c for interrupt handler vectors	CRB-VEC	Channel request block-interrupt transfer vector block
uba_driver		ADP	Adapter control block
uba_ctrl		ADP	ADP Extension
vbadata		ADP	ADP Extension, bus specific
swap_lw_bytes	Byte swap kernel routine	IOC\$VME_BYTE_SWAP_LONG	Swaps bytes of longword
swap_word_bytes	Byte swap kernel routine	IOC\$VME_BYTE_SWAP_WORD	Swaps bytes of a word
probe	Finds and checks status of device in system	Controller Initialization	
attach	Establishes communication with device	Unit Initialization	Prepares a device for operation
read	Reads data from a device	IO\$_READBLK	
write	Writes data to a device	IO\$_WRITEBLK	
physio	Perform I/O to/from user space kernel-support routine	\$QIO	Queue I/O request system service
start	Entry point start routine	STARTIO	Start I/O routine
open	Entry point open routine	\$ASSIGN	System service routine
close	Entry point close routine	\$DASSGN	System service routine

(continued on next page)

Table A-4 (Cont.) Driver Notions Porting from UNIX to VMS

UNIX Name	UNIX Description	VMS Name	VMS Description
intr	Entry point interrupt routine	INTERRUPT	Interrupt service routine
strategy	Entry point strategy routine	FDT	Function decision table QIO handling routine
config	System Configuration Utility	SYSGEN	System Generation Utility
SYSNAME	System configuration file		SYSGEN device table, ACF, and DDB

A.3.1.1.9 Assembling and Linking a VMEbus Driver Assemble the source files with the system's macro library (SYS\$LIBRARY:LIB.MLB) and include VMESUPPORT. For example:

```
$ MACRO QVDRIVER.MAR+SYS$LIBRARY:LIB.MLB/LIBRARY -
_$      +SYS$LIBRARY:VMESUPPORT/LIBRARY
```

Link the driver object file with the VMS global symbol table and the VME routines object library. The global symbol table is located in SYS\$SYSTEM and called SYS.STB and the VME routines are located in SYS\$SHARE:VME\$LIBRARY.OLB. If the driver consists of several source files, you must specify the file that contains the driver prologue table as the first file in the list. The linker options file must contain a BASE statement specifying a zero base for the executable image.

The following is an example of a LINK command used to link a VME device driver with the VME support routines:

```
$ CREATE QVDRIVER.OPT
BASE=0
[CtrlZ]
$ LINK /NOSYSSHR/NOTRACEBACK/NODEBUG/CONTIGUOUS QVDRIVER.OBJ, -
_$      SYS$SHARE:VME$LIBRARY/LIBRARY/SELECT, -
_$      QVDRIVER.OPT/OPTIONS, -
_$      SYS$SYSTEM:SYS.STB/SELECTIVE_SEARCH
```

The resulting image must consist of a single image section. The linker will report that the image has no transfer address; this report should be ignored.

Once you have linked or relinked a driver, copy its image file to the SYS\$LOADABLE_IMAGES directory. By default, the SYSGEN commands LOAD and CONNECT search for a driver in the SYS\$LOADABLE_IMAGES directory.

A.3.1.1.10 Loading a VME Device Driver You can load a VME device driver during the bootstrap program (for example, in SYSTARTUP.COM) or anytime after the system is booted. Note that you cannot autoconfigure VME devices.

To load the driver into system virtual memory, run the System Generation Utility (SYSGEN) from the system manager's account or from an account having CMKRN privilege. SYSGEN loads a VME device driver and creates the device's I/O data structures. For more details on loading a driver with SYSGEN, refer to the *VMS Device Support Manual*.

Invoke SYSGEN by entering the following command:

```
$ RUN SYS$SYSTEM:SYSGEN
```

SYSGEN responds with the following prompt and waits for further input:

```
SYSGEN>
```


VMS Version 5.4-3 Features

A.3 VMS Version 5.4-3 Programming Features

Use the **CONNECT** command (of **SYSGEN**) to load the driver and create the device's I/O database. You must specify the device name, the nexus number or decimal number of the VMEbus adapter, the VME address space CSR offset, and the interrupt vector offset.

The CSR offset is a full word. The offset allows a CSR to fall anywhere in the first 64K of VME address space. The interrupt vector is a byte offset with offsets up to 256 bytes. These vector offsets must be longword aligned.

You can obtain the adapter nexus number for the XMI-to-VME adapter by issuing the following **SHOW/ADAPTER** command:

```
SYSGEN> SHOW/ADAPTER
```

```
CPU Type: VAX 6000-530
```

Nexus (decimal)	Generic Name or Description
0010 16	XMI - 6000-500 processor
0020 32	XMI - 6000-500 processor
0040 64	XMI - memory module
0070 112	XMI - memory module
00A0 160	XMI - Disk/Tape Adapter (KDM70)
00C0 192	XMI - VME adapter
00D0 208	XMI - NI Adapter (DEMNA)

The **SHOW/BUS** command can also be used to list nexus numbers:

```
SYSGEN> SHOW/BUS
```

Cpu Type: VAX 6000-530	Cpu Connection: XMI
Bus Node Generic Name	Nexus(hex) Connection Address
XMI 00 01 XMI - 6000-500 processor	0010
XMI 00 02 XMI - 6000-500 processor	0020
XMI 00 04 XMI - memory module	0040
XMI 00 07 XMI - memory module	0070
XMI 00 0A XMI - Disk/Tape Adapter KDM70	00A0
XMI 00 0C XMI - VME adapter	00C0
XMI 00 0D XMI - NI adapter (DEMNA)	00D0

The following example illustrates how the **CONNECT** command is used:

```
SYSGEN> CONNECT QVA0/ADAPTER=192/CSR=%X9000 -  
SYSGEN> /DRIVER=QVDRIVER/VECTOR=%XB0
```

This command loads the driver **QVDRIVER**, if it is not already loaded, and creates the data structures (**DDB**, **CRB**, **IDB**, and **UCB**) needed to describe **QVA0**. It also causes the driver's controller and unit initialization routines to be executed. **QVA0** is the device name and number (**QV**=customer VME, **A0**=device #0). Note that Digital reserves driver names beginning with the letters **J** and **Q** to its customers.

The previous example specifies a driver that has its CSRs beginning at address **9000₁₆** of VME **A16** word-access space. The example also shows an interrupt vector of **B0₁₆**. Upon a VME interrupt, VME devices generate a status byte that can contain a vector value between **00₁₆** to **FC₁₆**.

A.3.1.1.11 VMS Macros Invoked by VME Drivers This section describes the VMS macros used by VME device drivers.

The VME macros are defined in **SYS\$LIBRARY:VMESUPPORT.MLB**. General information about the structure of macros and their arguments appears in the *VAX MACRO and Instruction Set Reference Manual*.

SWAPLONG

Swaps the bytes within each longword supplied.

Format

SWAPLONG longword

Parameters

longword

The address of the longword data that requires the bytes to be swapped.

Description

When passing a data word between a host CPU and a device with a differing byte-order pattern (big-endian and little-endian devices), the byte positions must be swapped. The SWAPLONG macro reads the location of the 4-byte data supplied in the longword argument and modifies the byte positions to a mirrored order.

SWAPWORD

Swaps the bytes within each word supplied.

Format

SWAPWORD word

Parameters

word

The address of the data (2 bytes) that requires the bytes to be swapped.

Description

When passing a data word between a host CPU and a device with a differing byte-order pattern (big-endian and little-endian devices), the byte positions must be swapped. The SWAPWORD macro reads the location of the 2-byte data supplied in the word argument and swaps the byte positions.

A.3.1.2 VME Driver Operating System Routines

This section describes the VMS operating system routines that are used by VME device drivers supporting the XMI-to-VME bus connection (DWMVA adapter). The routines provide DMA mapping, PIO mapping, and byte-swap manipulation for big- and little-endian support.

IOC\$ALOVMEMAP_DMA, IOC\$ALOVMEMAP_DMAN

Allocates a set of VME DMA map registers.

Module

[DRIVER]VMEDMA_XMI

Input

Location	Contents
UCB\$W_CRB	Address of CRB
CRB\$L_INTD+VEC\$L_ADP	Address of ADP
ADP\$W_MRNREGARY, ADP\$W_MRFREGARY, ADP\$L_MRACTMDRS	Map register descriptor arrays
For IOC\$ALOVMEMAP_DMA only	
R5	Address of UCB
UCB\$W_BCNT	The transfer byte count
UCB\$W_BOFF	Byte offset to start of transfer in first page
For IOC\$ALOVMEMAP_DMAN only	
R1	Address of the map register descriptor (VME_MD)
R2	Address of ADP
R3	Number of map registers to be allocated

Output

Location	Contents
R0	SS\$_NORMAL or SS\$_INSFMAPREG
R2	Address of ADP
ADP\$W_MRNREGARY, ADP\$W_MRFREGARY, ADP\$L_MRACTMDRS	Updated
For IOC\$ALOVMEMAP_DMA only	
R1	Destroyed
CRB\$L_INTD+VEC\$B_NUMREG	Number of map registers allocated
CRB\$L_INTD+VEC\$W_MAPREG	Starting map register number
For IOC\$ALOVMEMAP_DMAN only	
R1	Address of the map register descriptor (VME_MD)

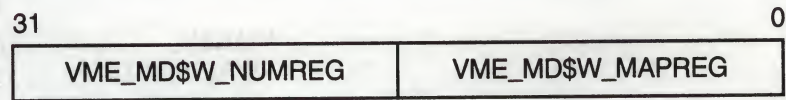
Synchronization

The caller of IOC\$ALOVMEMAP_DMA or IOC\$ALOVMEMAP_DMAN must be executing at fork IPL or above and must hold the corresponding fork lock in a VMS multiprocessor environment. Either routine returns control to its caller and the caller's IPL. The caller retains any spin locks it held at the time of the call.

Description

IOC\$ALOVMEMAP_DMA and IOC\$ALOVMEMAP_DMAN allocate a contiguous set of VME DMA map registers. IOC\$ALOVMEMAP_DMA records the allocation in the ADP and CRB (or in a map register descriptor using IOC\$ALOVMEMAP_DMAN). Figure A-6 shows the structure of the map register descriptor used by IOC\$ALOVMEMAP_DMAN.

Figure A-6 VME Map Register Descriptor (VME_MD)



ZK-3732A-GE

These routines differ in the way in which they determine the number of map registers they allocate:

- IOC\$ALOVMEMAP_DMA calculates the number of needed map registers using the values contained in UCB\$W_BCNT and UCB\$W_BOFF.
- IOC\$ALOVMEMAP_DMAN uses the value in R3 as the number of required registers.

If there are not enough contiguous map registers available, the routine returns an error status of SS\$_INSFMAPREG to its caller.

The caller of IOC\$ALOVMEMAP_DMAN must keep track of the map registers allocated because they eventually must be released. Care should be exercised in the consumption and management of map register resources.

Note that (when using the IOC\$ALOVMEMAP_DMA routine) if there are not enough map registers available, your driver has the option to put a fork block onto the map register allocation wait queue in the ADP. When registers are released, the release routine checks for waiting fork threads. If any are waiting, it will attempt to complete the allocation at that time.

IOC\$LOADVMEMAP_DMA, IOC\$LOADVMEMAP_DMAN

Loads a set of VME map registers for DMA.

Module

[DRIVER]VMEDMA_XMI

Input

Location	Contents
R0	VMEbus control flags:
	VME\$V_ RMWMODE Translate VME read-modify-write into XMI interlocked accesses
	VME\$V_ SWAPWORD See the adapter technical manual for details
	VME\$V_ SWAPLONG See the adapter technical manual for details
CRB\$L_INTD+VEC\$L_ADP	Address of ADP
For IOC\$LOADVMEMAP_DMA only	
R5	Address of the UCB
UCB\$W_BCNT	Number of bytes in transfer
UCB\$W_BOFF	Byte offset to start of transfer in first page
UCB\$L_SVAPTE	System virtual address of PTE for first page of transfer
UCB\$L_CRB	Address of CRB
CRB\$L_INTD+VEC\$B_NUMREG	Number of map registers allocated
CRB\$L_INTD+VEC\$W_MAPREG	Number of first map register allocated
UCB\$L_SVAPTE	System virtual address of PTE for the first page of the transfer
For IOC\$LOADVMEMAP_DMAN only	
R1	Address of the map register descriptor (VME_MD shown in Figure A-6)
R2	Address of ADP
R3	System virtual address (SVAPTE) of first page to transfer
R4	Byte count of the transfer
R5	Byte offset to start of transfer in first page

Output

Location	Contents
R0, R1, R2	Destroyed

Synchronization

A driver fork process calls IOC\$LOADVMEMAP_DMA or IOC\$LOADVMEMAP_DMAN at fork IPL, holding the corresponding fork lock in a VMS multiprocessor environment. Either routine returns control to its caller at the caller's IPL. The caller retains any spin locks it held at the time of the call.

Description

A driver fork process calls IOC\$LOADVMEMAP_DMA or IOC\$LOADVMEMAP_DMAN to load a previously allocated set of DMA map registers with page-frame numbers (PFNs). This enables a device to perform DMA transfer to or from the buffer indicated by the contents of UCB\$L_SVAPTE, UCB\$W_BCNT, and UCB\$W_BOFF (or contents of R3, R4, and R5 when using IOC\$LOADVMEMAP_DMAN).

Either IOC\$LOADVMEMAP_DMA or IOC\$LOADVMEMAP_DMAN confirms that sufficient map registers were previously allocated. If not, it issues a UBMAPEXCED bugcheck. Otherwise, it loads the appropriate PFN into each map register.

IOC\$LOADVMEMAP_DMA and IOC\$LOADVMEMAP_DMAN check the VMEbus control-flags register and set the appropriate bits in each map register.

The IOC\$ALOVMEAP_DMA routine loads and sets the mapping register valid for the number of mapping registers needed for the length of the DMA request. The routine sets the byte swapping requested and the type of access for the VME bus. Access type is whether VME read-modify-write operations are translated into XMI interlocked accesses or not.

IOC\$RELVMEMAP_DMA, IOC\$RELVMEMAP_DMAN

Releases a set of VME DMA map registers.

Module

[DRIVER]VMEDMA_XMI

Input

Location	Contents
ADP\$L_MRQFL	Head of queue of UCBs waiting for map registers
ADP\$W_MRNREGARY, ADP\$W_MRFREGARY, ADP\$L_MRACTMDRS	Map register descriptor arrays
For IOC\$RELVMEMAP_DMA only	
R5	Address of UCB
UCB\$L_CRB	Address of CRB
CRB\$L_INTD+VEC\$L_ADP	Address of ADP
CRB\$L_INTD+VEC\$B_NUMREG	Number of allocated map registers
For IOC\$RELVMEMAP_DMAN only	
R1	Address of map register descriptor (VME_MD shown in Figure A-6)
R2	Address of ADP

Output

Location	Contents
R0	SS\$_NORMAL or SS\$_SSFAIL
R1, R2	Destroyed
ADP\$W_MRNREGARY, ADP\$W_MRFREGARY, ADP\$L_MRACTMDRS	Updated

Synchronization

A driver fork process calls IOC\$RELVMEMAP_DMA or IOC\$RELVMEMAP_DMAN at fork IPL, holding the corresponding fork lock in a VMS multiprocessor environment.

Description

A driver fork process calls IOC\$RELVMEMAP_DMA or IOC\$RELVMEMAP_DMAN to release a previously allocated set of VME DMA map registers.

IOC\$RELVMEMAP_DMA obtains the location and number of the allocated map registers from CRB\$L_INTED+VEC\$W_MAPREG and CRB\$L_INTED+VEC\$B_NUMREG, respectively.

After adjusting the map register descriptor arrays, IOC\$RELVMEMAP_DMA examines the VME DMA-map-register wait queue. If the queue is empty, IOC\$RELVMEMAP_DMA returns successfully to its caller. If the queue contains waiting fork processes, IOC\$RELVMEMAP_DMA dequeues the first process and calls IOC\$ALOVMEMAP_DMA to attempt to allocate the set of map registers it requires.

When using the IOC\$ALOVMEMAP_DMA routine, if there are sufficient map registers, IOC\$RELVMEMAP_DMA restores R3 through R5 to the process and reactivates it. When this fork process returns control to IOC\$RELVMEMAP_DMA, IOC\$RELVMEMAP_DMA attempts to allocate map registers to the next waiting fork process. IOC\$RELVMEMAP_DMA continues to allocate map registers in this manner until the map-register wait queue is empty or it cannot satisfy the requirements of the process at the head of the queue. In the latter event, IOC\$RELVMEMAP_DMA reinserts the fork process's UCB in the queue and returns successfully to its caller.

IOC\$ALOVMEMAP_PIO

Allocates a set of VME PIO map registers.

Module

[DRIVER]VMEPIO_XMI

Input

Location

R3

UCB\$L_CRB

CRB\$L_INTD+VEC\$L_ADP

ADP\$W_MR2NREGAR,

ADP\$W_MR2FREGAR,

ADP\$L_MR2ACTMDR

Contents

Number of map registers to allocate

Address of CRB

Address of ADP

Map register descriptor arrays

Output

Location

R0

R1

R2

CRB\$L_INTD+VEC\$B_

NUMALT

ADP\$W_MR2NREGAR,

ADP\$W_MR2FREGAR,

ADP\$L_MR2ACTMDR

Contents

SS\$_NORMAL or SS\$_INSFMAPREG

Destroyed

Address of ADP

Number of map registers allocated

Updated

Synchronization

The caller of IOC\$ALOVMEMAP_PIO must be executing at fork IPL or above and must hold the corresponding fork lock in a VMS multiprocessor environment. IOC\$ALOVMEMAP_PIO returns control to its caller at the caller's IPL. The caller retains any spin locks it held at the time of the call.

Description

IOC\$ALOVMEMAP_PIO allocates a contiguous set of VME PIO map registers and records the allocation in the VMEbus adapter ADP and CRB.

IOC\$ALOVMEMAP_PIO searches the map register descriptor arrays for the required number of map registers. If there are not enough contiguous map registers available, the routine returns an error status of zero to its caller.

IOC\$LOADVMEMAP_PIO

Loads a set of VME PIO map registers.

Module

[DRIVER]VMEPIO_XMI

Input

Location	Contents
R0	VME page-frame numbers (PFNs)
R1	VMEbus access flags:
	VME\$V_A16 VME access in short address-space mode
	VME\$V_A24 VME access in standard address-space mode
	VME\$V_A32 VME access in extended address-space mode
	VME\$V_BYTE VME byte accesses
	VME\$V_WORD VME word accesses
	VME\$V_LONG VME longword accesses
R3	Number of registers to load
R5	Address of UCB
UCB\$L_CRB	Address of CRB
CRB\$L_INTD+VEC\$W_NUMALT	Number of PIO map registers allocated
CRB\$L_INTD+VEC\$W_MAPALT	Number of first PIO map register allocated
CRB\$L_INTD+VEC\$L_ADP	Address of ADP
ADP\$L_MR2ADDR	Address of first VME PIO map register

Output

Location	Contents
R0	SS\$_NORMAL, SS\$_INSFMAPREG, or SS\$_FAIL
R1, R2	Destroyed

Synchronization

A driver fork process calls IOC\$LOADVMEMAP_PIO at fork IPL, holding the corresponding fork lock in a VMS multiprocessor environment. IOC\$LOADVMEMAP_PIO returns control to its caller at the caller's IPL. The caller retains any spin locks it held at the time of the call.

VME Driver Operating System Routines

IOC\$LOADVMEMAP_PIO

Description

A driver fork process calls IOC\$LOADVMEMAP_PIO to load a previously allocated set of map registers with VME PFNs. For the DWMVA adapter, a VME PFN for programmed I/O access contains bits A31:A20. The low-order bits A19:A0 are taken from the XMI I/O address offset that corresponds to the map register in question. For more details, see the adapter technical manual.

The VME address type, access length, and access mode are all controlled by setting or clearing the appropriate flags in the access flags register.

IOC\$LOADVMEMAP_PIO confirms that sufficient VME PIO map registers have been previously allocated. If not, it issues a UBMAPEXCED bugcheck. Otherwise, it loads the appropriate PFN into each map register and sets the map register valid bit.

IOC\$RELVMEMAP_PIO

Releases a set of VME PIO map registers.

Module

[DRIVER]VMEPIO_XMI

Input

Location	Contents
R5	Address of UCB
UCB\$L_CRB	Address of CRB
CRB\$L_INTD+VEC\$L_ADP	Address of ADP
CRB\$L_INTD+VEC\$B_NUMALT	Number of allocated PIO map registers
ADP\$L_MR2QFL	Head of queue of UCBs waiting for PIO map registers
ADP\$W_MR2NREGAR, ADP\$W_MR2FREGAR, ADP\$L_MR2ACTMDR	PIO map register descriptor arrays

Output

Location	Contents
R0	SS\$_NORMAL or SS\$_SSFAIL
R1, R2	Destroyed
ADP\$W_MR2NREGAR, ADP\$W_MR2FREGAR, ADP\$L_MR2ACTMDR	Updated

Synchronization

A driver fork process calls IOC\$RELVMEMAP_PIO at fork IPL, holding the corresponding fork lock in a VMS multiprocessor environment.

Description

A driver fork process calls IOC\$RELVMEMAP_PIO to release a previously allocated set of VME PIO map registers in the ADP.

IOC\$RELVMEMAP_PIO obtains the location and number of the allocated map registers from CRB\$L_INTD+VEC\$W_MAPALT and CRB\$L_INTD+VEC\$W_NUMALT, respectively.

After adjusting the PIO map register descriptor arrays, IOC\$RELVMEMAP_PIO examines the VME PIO-map-register wait queue. If the queue is empty, IOC\$RELVMEMAP_PIO returns successfully to its caller. If the queue contains waiting fork processes, IOC\$RELVMEMAP_PIO dequeues the first process and calls IOC\$ALOVMEAP_PIO to attempt to allocate the set of map registers it requires.

VME Driver Operating System Routines

IOC\$RELVMEMAP_PIO

If there are sufficient alternate map registers, IOC\$RELVMEMAP_PIO restores R3 through R5 to the process and reactivates it. When this fork process returns control to IOC\$RELVMEMAP_PIO, IOC\$RELVMEMAP_PIO attempts to allocate map registers to the next waiting fork process. IOC\$RELVMEMAP_PIO continues to allocate map registers in this manner until the VMEPIO-map-register wait queue is empty or it cannot satisfy the requirements of the process at the head of the queue. In the latter event, IOC\$RELVMEMAP_PIO reinserts the fork process's UCB in the queue and returns successfully to its caller.

IOC\$VME_BYTE_SWAP_LONG

Swaps the bytes within each longword in a given data transfer buffer.

Module

[DRIVER]VME_SUPPORT

Input

Location

R0

Contents

Length of the data transfer buffer in bytes.
This number should fall on a longword boundary.

R1

Address of the data transfer buffer.

Output

Location

R0, R1

Contents

Destroyed

(All other registers preserved)

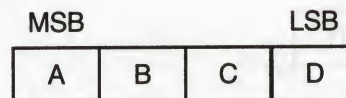
Synchronization

A driver calls IOC\$VME_BYTE_SWAP_LONG in kernel mode at or above IPL 2.

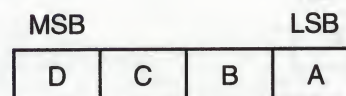
Description

IOC\$VME_BYTE_SWAP_LONG swaps the bytes within each longword of a given data transfer. The data is read from an input system buffer, then the byte positions of each longword are modified to a mirrored order, as shown in the following figure:

Original Format:



Swapped Format:



ZK-3733A-GE

Note that if the byte length of the buffer is not an exact number of longwords, the bytes in the last incomplete longword are unaffected.

IOC\$VME_BYTE_SWAP_WORD

Swaps the bytes within each word in a given data transfer buffer.

Module

[DRIVER]VME_SUPPORT

Input

Location

R0

R1

Contents

Length of the data transfer buffer in bytes.
This number should fall on a word boundary.

Address of the data transfer buffer.

Output

Location

R0, R1

Contents

Destroyed

(All other registers preserved)

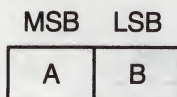
Synchronization

A driver calls IOC\$VME_BYTE_SWAP_WORD in kernel mode at or above IPL 2.

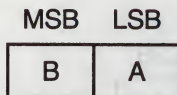
Description

IOC\$VME_BYTE_SWAP_WORD swaps the bytes within each word of a given data transfer. The data is read from an input system buffer, then the byte positions of each word are modified to a mirrored order, as shown in the following figure:

Original Format:



Swapped Format:



ZK-3734A-GE

Note that if the buffer contains an odd number of bytes, the last byte in the incomplete word at the end of the buffer is unaffected.

A.3.1.3 Sample Driver for a VMEbus DR11-W

The following sample driver controls the Ikon DR11-W Emulator featuring a DMA interface for a VMEbus device. Table A-5 outlines the driver code by listing the sections and routines in order of their occurrence. You can obtain a machine-readable copy of this driver from SYS\$EXAMPLES:QKDRIVER.MAR.

Table A-5 DR11-W VME Driver Code Contents

Driver Code Points	Function
① External symbols	Defined
② Local symbols	Defined
③ Argument list (AP)	Defined for device-dependent QIO parameters
④ Constants	Defined
⑤ Device-specific UCB fields	Defined
⑥ Device-register offsets from CSR	Defined
⑦ Bit positions of CSR	Defined
⑧ Driver prologue table (DPT)	Initialized with DPT_STORE
⑨ Driver dispatch table (DDT)	Initialized with DDTAB
⑩ Function decision table (FDT)	Loaded with FUNCTAB
⑪ QK_CONTROL_INIT routine	For controller initialization
⑫ Byte swap macro (SWAPWORD)	Called
⑬ QK_READ_WRITE FDT routine	For data transfers servicing READLBLK, READVBLK, READPBLK, WRITELBLK, WRITEVBLK, and WRITEPBLK
⑭ QK_START routine	Starting an I/O transfer
⑮ QK_TIME_OUT routine	Handling a DR11-W device timeout
⑯ QK_INTERRUPT routine	Handling interrupts generated by the DR11-W
⑰ QK_REGISTER routine	Handling DR11-W CSR transfers
⑱ QK_CANCEL routine	Canceling an I/O operation
⑲ QK_DEV_RESET routine	Performing a device reset

VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```
.TITLE QKDRIVER - VAX/VMS VMEbus QKon DR11-W Emulator DRIVER
.IDENT 'X-01'
```

```
;
;*****
; *
; * COPYRIGHT (c) 1990 BY
; * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
; * ALL RIGHTS RESERVED.
; *
; * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
; * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
; * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
; * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
; * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
; * TRANSFERRED.
; *
; * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
; * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
; * CORPORATION.
; *
; * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
; * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
; *
; *****
;
; ++
;
; FACILITY:
;
; VAX/VMS Executive, I/O Drivers
;
; ABSTRACT:
;
; This module contains the driver for the VMEbus Ikon DR11-W Emulator
; (XMI).
;
; ENVIRONMENT:
;
; Kernel Mode, Non-paged
;
; AUTHOR:
;
; --
;
; .SBTTL External and local symbol definitions
;
; 1
; External symbols
```



```

$ACBDEF      ; AST control block
$ADPDEF      ; Adapter control block
$CRBDEF      ; Channel request block
$DCDEF       ; Device types
$DDBDEF      ; Device data block
$DEVDEF      ; Device characteristics
$DPTDEF      ; Driver prolog table
$DYNDEF      ; Dynamic data structure types
$EMBDEF      ; EMB offsets
$IDBDEF      ; Interrupt data block
$IODEF       ; I/O function codes
$IPLDEF      ; Hardware IPL definitions
$IRPDEF      ; I/O request packet
$PRDEF       ; Internal processor registers
$PRIDEF      ; Scheduler priority increments
$SSDEF       ; System status codes
$UCBDEF      ; Unit control block
$VECDEF      ; Interrupt vector block
$XADEF       ; Define device specific characteristics
$XVIBDEF     ; VME definitions

```

②

```

; Local symbols
; (Your local symbols here)
; Argument list (AP) offsets for device-dependent QIO parameters

```

③

```

P1      = 0      ; First QIO parameter
P2      = 4      ; Second QIO parameter
P3      = 8      ; Third QIO parameter
P4      = 12     ; Fourth QIO parameter
P5      = 16     ; Fifth QIO parameter
P6      = 20     ; Sixth QIO parameter

```

④

```

; Other constants

QK_DMA_DEF_TIMEOUT      = 10      ; 10 second DMA default timeout
QK_READ_SYNCH_TIMEOUT   = 10      ; 10 second Time out to synchronize
                                ; with a READ request.
QK_DEF_BUFSIZ           = 65535   ; Default buffer size
QK_RESET_DELAY          = <<2+9>/10> ; Delay N microseconds after RESET
                                ; (rounded up to 10 microsec intervals)
QK_ADDR_MOD_10089       = ^XD00   ; Select 32 bit addressing on the VME.
                                ; Which is 0D. This value is in the
                                ; high byte of the Register.

QK_ADDR_MOD_10099       = ^X8B00  ; Block Mode.

```

⑤

```

; DR11-W definitions that follow the standard UCB fields

```


VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```

$DEFINI UCB
.=UCB$L_DPC+4
$DEF UCB$L_MAPREG_DESC ; The Mapping Register Descriptor.
$DEF UCB$W_START_MAPREG ; The Starting Map Register.
    .BLKW 1
$DEF UCB$W_NUMBER_MAPREG ; The number of Map Registers.
    .BLKW 1
$DEF UCB$W_QK_UNEXPECTED ; Counter for # of unexpected interrupts.
    .BLKW 1
$DEF UCB$W_QK_CSRTMP ; Temporary storage of Control Reg image
    .BLKW 1
$DEF UCB$W_QK_BARTMPLOW ; Temporary storage of BAR Reg LOW image
    .BLKW 1
$DEF UCB$W_QK_BARTMPHIGH ; Temporary storage of BAR Reg HIGH image
    .BLKW 1
$DEF UCB$W_QK_WCRTMPLOW ; Temporary storage of WCR Reg LOW image
    .BLKW 1
$DEF UCB$W_QK_WCRTMPHIGH ; Temporary storage of WCR Reg HIGH image.
    .BLKW 1
$DEF UCB$W_QK_PULSE ; Storage for the Pulse command register.
    .BLKW 1
$DEF UCB$W_QK_VECTOR ; Storage for the Vector and Address
    .BLKW 1 ; Modifier Register.
$DEF UCB$W_QK_CSR ; Saved STATUS Reg on interrupt
    .BLKW 1
$DEF UCB$W_QK_BARLOW ; Saved BAR register LOW on interrupt
    .BLKW 1
$DEF UCB$W_QK_BARHIGH ; Saved BAR register HIGH on interrupt
    .BLKW 1
$DEF UCB$W_QK_WCRLOW ; Saved WCR register LOW on interrupt
    .BLKW 1
$DEF UCB$W_QK_WCRHIGH ; Saved WCR register HIGH on interrupt
    .BLKW 1
$DEF UCB$W_QK_ERROR ; Saved Error return.
    .BLKW 1

; Bit positions for device-dependent status field in UCB
$VIELD UCB,0,<- ; UCB device specific bit definitions
    <READ_READY,,M>,- ; The READ partner QIO is ready.
    <WAITING_FOR_READ,,M>,- ; The Waiting for READ partner interrupt.
>
UCB$K_SIZE=.
$DEFEND UCB

```

6

; Device register offsets from CSR address

```

$DEFINI QK ; Start of Ikon DR11-W definitions
$DEF QK_CONTROL ; Control Register
$DEF QK_STATUS ; Status Register
    .BLKW 1

$DEF QK_DATA_OUT ; Data Out Register
$DEF QK_DATA_IN ; Data In Register
    .BLKW 1

$DEF QK_MODIFIER_VECTOR ; Address Modifier and Vector Register.
    .BLKW 1

$DEF QK_PULSE_COMMAND ; Pulse Command Register
    .BLKW 1
    .BLKW 5 ; Empty space in register area.

$DEF QK_BAR_LOW_WRITE ; DMA address Low 16 bits. WRITE
    .BLKW 1

```


VMS Version 5.4-3 Features Sample Driver for a VMEbus DR11-W

```

$DEF    QK_WCR_LOW                      ; DMA Word Count Low 16 bits register.
                                .BLKW    1
$DEF    QK_BAR_LOW_READ                 ; DMA address Low 16 bits. READ
                                .BLKW    1
                                .BLKW    1 ; Empty space in register area.
$DEF    QK_BAR_HIGH_WRITE               ; DMA address High 16 bits. Write
                                .BLKW    1
$DEF    QK_WCR_HIGH                     ; DMA Word Count High 16 bits register.
                                .BLKW    1
$DEF    QK_BAR_HIGH_READ                ; DMA address High 16 bits. READ
                                .BLKW    1

```

7

; Bit positions for device control/status register

```

$EQLST QK$K_,0,1,<-                ; Define CSR FNCT bit values
    <FNCT1,2>-
    <FNCT2,4>-
    <FNCT3,8>-
    <STATUSA,2048>-                ; Define CSR STATUS bit values
    <STATUSB,1024>-
    <STATUSC,512>-
>
$VIELD QK_CONTROL,0,<-            ; Control register
    <GO,,M>,-                      ; Start device
    <FNCT,3,M>,-                  ; CSR FNCT bits
    <SDIR,,M>,-                  ; Software direction
    <UNUSED1,,M>,-              ; Unused bit
    <IE,,M>,-                    ; Enable interrupts
    <TERM,,M>,-                  ; Terminate active DMA.
    <CYCLE,,M>,-                ; Starts slave transmit
    <UNUSED2,3,M>,-             ; UNUSED bits
    <MCLR,,M>,-                 ; Master Clear.
    <RPER,,M>,-                 ; Reset Parity Error Flag.
    <RATN,,M>,-                 ; Reset Attention flag and its interrupt.
    <RDMA,,M>,-                 ; Reset DMA Done flag and its interrupt.
>
$VIELD QK_STATUS,0,<-            ; Status register
    <DFLG,,M>,-                  ; Device Flag
    <FNCT,3,M>,-                ; FNCT bits
    <SDIR,1,M>,-                ; State of SDIR latch
    <BERR,1,M>,-                ; Bus error flag
    <IE,,M>,-                   ; Enable interrupts
    <READY,,M>,-                ; DMA Ready.
    <UNUSED1,,M>,-              ; UNUSED bit
    <STATUS,3,M>,-              ; Status bits
    <PERR,,M>,-                 ; Parity error flag.
    <ATTN,,M>,-                 ; State of Attention H input.
    <ATTF,,M>,-                 ; Attention interrupt.
    <DMAF,,M>,-                 ; DMA Done interrupt.
>

```

```

$DEFEND QK                      ; End of DR11-W definition

```

8

.SBTTL Device Driver Tables

; Driver prologue table

VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```

DPTAB      -                ; DPT-creation macro
            END=QK_END,-    ; End of driver label
            ADAPTER=VME,-   ; Adapter type
            FLAGS=DPT$M_SVP,- ; Allocate system page table
            UCBSIZE=UCB$K_SIZE,- ; UCB size
            NAME=QKDRIVER   ; Driver name
DPT_STORE INIT                ; Start of load
                                ; initialization table
DPT_STORE UCB,UCB$B_FLCK,B,SPL$C_IOLOCK8 ; Device fork IPL
DPT_STORE UCB,UCB$B_DIPL,B,22    ; Device interrupt IPL
DPT_STORE UCB,UCB$L_DEVCHAR,L,<-  ; Device characteristics
            DEV$M_AVL!-        ; Available
            DEV$M_RTM!-        ; Real Time device
            DEV$M_ELG!-        ; Error Logging enabled
            DEV$M_IDV!-        ; input device
            DEV$M_ODV>         ; output device
DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$REALTIME ; Device class
DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$XVIB ; Device Type
DPT_STORE UCB,UCB$W_DEVBUFSIZ,W,- ; Default buffer size
            QK_DEF_BUFSIZ
DPT_STORE REINIT                ; Start of reload
                                ; initialization table
DPT_STORE DDB,DDB$L_DDT,D,QK$DDT ; Address of DDT
DPT_STORE CRB,CRB$L_INTD+4,D,-   ; Address of interrupt
            QK_INTERRUPT        ; service routine
DPT_STORE CRB,CRB$L_INTD+VEC$L_INITIAL,- ; Address of controller
            D,QK_CONTROL_INIT   ; initialization routine
DPT_STORE END                    ; End of initialization
                                ; tables

```

9

; Driver dispatch table

```

DDTAB      -                ; DDT-creation macro
            DEVNAM=QK,-      ; Name of device
            START=QK_START,- ; Start I/O routine
            FUNCTB=QK_FUNCTABLE,- ; FDT address
            CANCEL=QK_CANCEL ; Cancel I/O routine

```

10

; Function dispatch table

```

QK_FUNCTABLE:                ; FDT for driver
    FUNCTAB ,-                ; Valid I/O functions
        <READPBLK,READLBLK,READVBLK,WRITEPBLK,WRITELBLK,WRITEVBLK>
    FUNCTAB ,                  ; No buffered functions
    FUNCTAB QK_READ_WRITE,-    ; Device-specific FDT
        <READPBLK,READLBLK,READVBLK,WRITEPBLK,WRITELBLK,WRITEVBLK>
    FUNCTAB +EXE$QIODRVPKT,-   ;
        <READPBLK,READLBLK,READVBLK,WRITEPBLK,WRITELBLK,WRITEVBLK>

```

11

.SBTTL QK_CONTROL_INIT, Controller initialization


```

; ++
; QK_CONTROL_INIT, Called when driver is loaded, system is booted, or
; power failure recovery.
;
; Functional Description:
;
;     1) Allocates the direct data path permanently
;     2) Assigns the controller data channel permanently
;     3) Clears the Control and Status Register
;     4) If power recovery, requests device time-out
;
; Inputs:
;
;     R4 = address of CSR
;     R5 = address of IDB
;     R6 = address of DDB
;     R8 = address of CRB
;
; Outputs:
;
;
; --

QK_CONTROL_INIT:
    JSB      G^INI$BRK
    MOVL     IDB$L_UCBLST(R5),R0      ; Address of UCB
    MOVL     R0,IDB$L_OWNER(R5)      ; Make permanent controller owner
    BISW     #UCB$M_ONLINE,UCB$W_STS(R0) ; Set device status "on-line"

    CLRW     UCB$W_QK_UNEXPECTED(R0) ; Init Unexpected Interrupt counter.

10$:  PUSHR   #^M<R3,R5>              ; Save R5
      MOVZBL IDB$B_VECTOR(R5),R1      ; Get the vector address.
      ROTL   #2,R1,R1                ; Normalize the vector
      MOVZWL QK_STATUS(R4),R2         ; Read the CSR.

12:   SWAPWORD R2                    ; Swap the bytes.
      MOVL   #QK_ADDR_MOD_10089,R3    ; Set R3 to the Address Modifier value.
      BBC    #QK_STATUS$V_DFLG,R2,50$; Branch if this is the 10089 revision.
      MOVL   #QK_ADDR_MOD_10099,R3    ; Set R2 to the Address Modifier value.

50$:  ADDL2   R3,R1                   ; Add in the Address Modifier.
      MOVW   R1,UCB$W_QK_VECTOR(R0)   ; Save the Vector and Address Mod value.
      SWAPWORD R1                    ; Swap the bytes.
      MOVW   R1,QK_MODIFIER_VECTOR(R4) ; Set the vector ID.

      MOVL   R0,R5                    ; Copy UCB address to R5
      BSBW   QK_DEV_HWRESET
      POPR   #^M<R3,R5>              ; Restore R5
      RSB
13:   .SBTTL  QK_READ_WRITE, FDT for device data transfers

```


VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```

; ++
; QK_READ_WRITE, FDT for READLBLK, READVBLK, READPBLK, WRITELBLK, WRITEVBLK,
; WRITEPBLK
;
; Functional description:
;
; 1) Rejects QUEUE I/O's with odd transfer count
;
; Inputs:
;
; R3 = Address of IRP
; R4 = Address of PCB
; R5 = Address of UCB
; R6 = Address of CCB
; R8 = Address of FDT routine
; AP = Address of P1
; P1 = Buffer Address
; P2 = Buffer size in bytes
; P3 = DMA Time Out Time in seconds
; P4 = VMEbus control flags.
;
; Outputs:
;
; R0 = Error status if odd transfer count
;
; --
QK_READ_WRITE:
    BLBS    P1(AP), 2$           ; The Buffer address must not be on
                                ; a byte boundary.
    BLBC    P2(AP), 20$          ; Branch if transfer count even
2$: MOVZWL  #SS$_BADPARAM, R0    ; Set error status code
5$: JMP     G^EXE$ABORTIO        ; Abort request

20$: TSTL   P2(AP)               ; Error if no transfer count.
    BEQL   2$

    MOVL   P3(AP), IRP$L_MEDIA(R3) ; Save the Time Out time.
    BNEQ   30$                  ; Branch if there is a time out time.
    MOVL   #QK_DMA_DEF_TIMEOUT, - ; Set Time Out time to the default.
    IRP$L_MEDIA(R3)

30$: MOVL   P4(AP), IRP$L_MEDIA+4(R3) ; Save the VMEbus control flags.

    MOVL   P1(AP), R0            ; Get the buffer address.
    MOVL   P2(AP), R1            ; Get the byte count.
    JSB    G^EXE$MODIFYLOCKR     ; Check buffer for access and lock down
    BLBC   R0, 5$
    RSB                                ; the buffer.

```

14

```
.SBTTL QK_START, Start I/O routines
;
; ++
; QK_START - Start a data transfer, set characteristics, enable ATTN AST.
;
; Functional Description:
;
; This routine has two major functions:
;
; 1) Start an I/O transfer. This transfer can be in either word
; or block mode. The FNCTN bits in the DR11-W CSR are set. If
; the transfer count is zero, the STATUS bits in the DR11-W CSR
; are read and the request completed.
;
; Inputs:
;
; R3 = Address of the I/O request packet
; R5 = Address of the UCB
;
; Outputs:
;
; R0 = final status and number of bytes transferred
; R1 = value of CSR STATUS bits and value of input data buffer register
; Device errors are logged
; Diagnostic buffer is filled
;
; --
;
; .ENABL LSB

QK_START:
    ASSUME IDB$$_CSR EQ 0
    MOVL   UCB$$_CRB(R5),R4          ; Address of CRB
    MOVL   @CRB$$_INTD+VEC$$_IDB(R4),R4 ; Get the CSR address.

    MOVAL  UCB$$_MAPREG_DESC(R5),R1 ; Set R1 to the address of mapreg desc.
    MOVL   UCB$$_CRB(R5),R2          ; Get CRB address.
    MOVL   CRB$$_INTD+VEC$$_ADP(R2),R2 ; Get address of ADP.

    PUSHL  R3                        ; Save R3.
    MOVL   IRP$$_BCNT(R3),R0          ; Get the byte count.
    MOVZWL IRP$$_BOFF(R3),R3
    MOVAB  ^X3FF(R0)[R3],R3          ; Calculate highest relative byte and round
    ASHL   #-9,R3,R3                 ; Calculate number of map registers required

    BSBW   IOC$$_ALOVMEMAP_DMAN       ; Allocate a set of VME map regs.
    POPL   R3                          ; Restore R3
    BLBS   R0,50$

    MOVZWL #SS$$_INSFMAPREG,R0        ; Set to error and end QIO.
    CLRL   R1
    JMP    QIO_DONE

50$:
    MOVL   IRP$$_MEDIA+4(R3),R0       ; Get the VMEbus control flags.
    MOVAL  UCB$$_MAPREG_DESC(R5),R1 ; Set the Mapreg desc address.
    PUSHR  ^M<R3,R4,R5>              ; Save R3-R5.
    MOVL   IRP$$_BCNT(R3),R4          ; Set R4 to the byte count.
    MOVZWL IRP$$_BOFF(R3),R5          ; Set R5 to the byte offset into 1st page.
    MOVL   IRP$$_SVAPTE(R3),R3        ; Set R3 to the SVAPTE of first page.
    BSBW   IOC$$_LOADVMEMAP_DMAN      ; Load the VME mapping registers.
    POPR   ^M<R3,R4,R5>              ; Restore R3-R5.
```


VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```

;
; Build the BAR registers.
;
    MOVZWL  IRP$W_BOFF(R3),R1      ; Byte offset in first page of xfer
    INSV    UCB$W_START_MAPREG(R5),#9,#16,R1
                                           ; Insert the Starting Map Register number
                                           ; R1 contains the BAR value.
    ASHL    #-1,R1,R1              ; The DR11-W wants the data shifted
                                           ; one place to the right.

    MOVW    R1,UCB$W_QK_BARTMPLOW(R5) ; Save the BAR Low Register value.

    ASHL    #-16,R1,R2             ; Set R1 to BAR High value.
    MOVW    R2,UCB$W_QK_BARTMPHIGH(R5) ; Save the BAR High Register value.

;
; Store the Word Count register contents.
;
    MOVL    IRP$L_BCNT(R3),R0      ; Fetch byte count
    ASHL    #-1,R0,R1             ; Make byte count into word count
    DECL    R1                    ; The Ikon DR11-2 wants # of words
                                           ; Minus 1 for the Word Count.
    ASHL    #-16,R1,R0            ; R1 Word has WC Low value.
                                           ; R0 Word has WC High value.

    MOVW    R1,UCB$W_QK_WCRTMPLOW(R5) ; Set the WC Low Register value.
    MOVW    R0,UCB$W_QK_WCRTMPHIGH(R5) ; Set the WC High Register value.

; Initialize the CSR contents for a Read. Enable interrupts and set the Go
; Bit. Set the 1st function bit to set direction
; Use the Pulse command Function 2 to interrupt the Transmitter partner.
;
    MOVW    #<QK_CONTROL$M_IE!QK_CONTROL$M_GO!QK$K_FNCT1>,-
    UCB$W_QK_CSRTMP(R5)
    MOVW    #QK$K_FNCT2,UCB$W_QK_PULSE(R5)

    DEVICELOCK -
        LOCKADDR=UCB$L_DLCK(R5),- ; Lock device access
        SAVIPL=-(SP),-           ; Save current IPL
        PRESERVE=NO              ; DON'T Preserve R0

; Branch if a Read request.
;
    CMPW    #IO$_READPBLK,IRP$W_FUNC(R3)
    BEQL    1000$

; Write Request. Make sure that the Read Partner is ready.
;
    CLRW    UCB$W_QK_PULSE(R5)
    BBS     #UCB$V_READ_READY,UCB$W_DEVSTS(R5),500$
    BISW    #UCB$M_WAITING_FOR_READ,-
    UCB$W_DEVSTS(R5)              ; Set the flag that we are waiting for
                                           ; the READ partner to be ready.

    WFIKPBH QK_TIME_OUT,#QK_READ_SYNCH_TIMEOUT ; Wait for Read ATTN interrupt
                                           ; indicating the READ partner
                                           ; is Ready.

    IOFORK

    DEVICELOCK -
        LOCKADDR=UCB$L_DLCK(R5),- ; Lock device access
        SAVIPL=-(SP),-           ; Save current IPL
        PRESERVE=NO              ; DON'T Preserve R0

    MOVL    UCB$L_IRP(R5),R3      ; Get the IRP.

```

VMS Version 5.4-3 Features Sample Driver for a VMEbus DR11-W

```

500$: BICW    #<UCB$M_READ_READY!UCB$M_WAITING_FOR_READ>,-
        UCB$W_DEVSTS(R5)          ; Clear the READ Ready Flag and Waiting
                                   ; For Read flag.

; Initialize the CSR contents for a WRITE. Enable interrupts, set the Go, and
; Cycle Bits.
;
        MOVW    #<QK_CONTROL$M_IE!QK_CONTROL$M_GO!QK_CONTROL$M_CYCLE>,-
        UCB$W_QK_CSRTMP(R5)

1000$:

        SETIPL  #31,-              ; Raise to IPL POWER
        ENVIRON=UNIPROCESSOR

        MOVW    UCB$W_QK_WCRTMPLOW(R5),R0 ; Get the WC low register.
        SWAPWORD R0                ; Swap the bytes.
        MOVW    R0,QK_WCR_LOW(R4)   ; Set the WC Low Reg.

        MOVW    UCB$W_QK_WCRTMPHIGH(R5),R0 ; Get the WC High register.
        SWAPWORD R0                ; Swap the bytes.
        MOVW    R0,QK_WCR_HIGH(R4)  ; Set the WC High Reg.

        MOVW    UCB$W_QK_BARTMPLOW(R5),R0 ; Set the Buffer Address Registers.
        SWAPWORD R0
        MOVW    R0,QK_BAR_LOW_WRITE(R4)
        MOVW    UCB$W_QK_BARTMPHIGH(R5),R0
        SWAPWORD R0
        MOVW    R0,QK_BAR_HIGH_WRITE(R4)

        CMPW    #IO$_READPBLK,IRP$W_FUNC(R3)
        BNEQ    1010$
        MOVW    UCB$W_QK_PULSE(R5),R0 ; Set the pulse command to set ATTN
        SWAPWORD R0                ; for Reads.
        MOVW    R0,QK_PULSE_COMMAND(R4)

1010$: MOVW    UCB$W_QK_CSRTMP(R5),R0 ; Move all bits to CSR
        SWAPWORD R0
        MOVW    R0,QK_CONTROL(R4)

; Wait for transfer complete interrupt, powerfail, or device time-out

        WFIKPBCH QK_TIME_OUT,IRP$L_MEDIA(R3) ; Wait for interrupt

; Device has interrupted, FORK

        IOFORK                                ; FORK to lower IPL

; Handle request completion, release VME resources, check for errors

        MOVZWL  #SS$_NORMAL,-(SP)            ; Assume success, store code on stack
        MOVAL   UCB$L_MAPREG_DESC(R5),R1 ; Get address of mapreg desc.
        MOVL    UCB$L_CRB(R5),R2           ; Get CRB address.
        MOVL    CRB$L_INTD+VEC$L_ADP(R2),R2 ; Get address of ADP.
        BSBW    IOC$RELVME_MAP_DMAN       ; Release the mapping registers.

; Check for errors and return status

        CMPW    UCB$W_QK_WCRHIGH(R5),#^XFFFF ; All words transferred?
        BNEQ    1080$                        ; NO
        CMPW    UCB$W_QK_WCRLow(R5),#^XFFFF ; All words transferred?
        BEQL    1100$                        ; Yes

1080$: MOVZWL  #SS$_OPINCOMPL,(SP)            ; No, flag operation not complete
        BICW    #<UCB$M_READ_READY!-
        UCB$M_WAITING_FOR_READ>,-
        UCB$W_DEVSTS(R5)          ; Clear the read ready flags.

```


VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```

1100$: BBC      #QK_STATUS$V_PERR,UCB$W_QK_CSR(R5),1110$ ; Branch on CSR error bit
1105$: MOVZWL   UCB$W_QK_ERROR(R5),(SP) ; Flag for controller/drive error status
      BSBW     QK_DEV_RESET           ; Reset DR11-W
      BRB      1200$
1110$: BBS      #QK_STATUS$V_BERR,UCB$W_QK_CSR(R5),1105$

1200$: MOVL     (SP)+,R0                ; Get final device status
      MOVZWL   UCB$W_QK_WCRLOW(R5),R1 ; Return Word Count.
      ASHL     #16,R1,R1
      MOVW     UCB$W_QK_CSR(R5),R1    ; Return CSR in IOSB

QIO_DONE:
      REQCOM                    ; Finish request in exec

>
      .PAGE
15      .SBTTL DR11-W DEVICE TIME-OUT
; ++
; DR11-W device TIME-OUT
; If a DMA transfer was in progress, release UBA resources.
; For DMA or WORD mode, deliver ATTN AST's, log a device timeout error,
; and do a hard reset on the controller.
;
; Clear DR11-W CSR
; Return error status
;
; Power failure will appear as a device time-out
; --
      .ENABL LSB
QK_TIME_OUT: ; Time-out for DMA transfer
      IOFORK                    ; Fork to complete request
      MOVVAL   UCB$L_MAPREG_DESC(R5),R1 ; Get address of mapreg desc.
      MOVL     UCB$L_CRB(R5),R2        ; Get CRB address.
      MOVL     CRB$L_INTD+VEC$L_ADP(R2),R2 ; Get address of ADP.
      BSBW     IOC$RELVEMAP_DMAN      ; Release the mapping registers.

      BSBW     QK_REGISTER            ; Read DR11-W registers
      BSBW     QK_DEV_RESET          ; Reset controller
      MOVZWL   #SS$_TIMEOUT,R0       ; Assume error status
      CLRL     R1
      BBC      #UCB$V_CANCEL,-
      UCB$W_STS(R5),20$              ; Branch if not cancel
      MOVZWL   #SS$_CANCEL,R0        ; Set status
20$: BBC      #UCB$V_WAITING_FOR_READ,-
      UCB$W_DEVSTS(R5),25$          ; Branch if waiting for Read.
      INCL     R1                    ; Set R1 to a 1 to indicate Waiting for
      ; Read.
      ; Clear unwanted flags.
25$: INSV     R1,#16,#16,R0          ; Insert the Time out type.
      MOVZWL   UCB$W_QK_WCRLOW(R5),R1 ;
      ASHL     #16,R1,R1
      MOVW     UCB$W_QK_CSR(R5),R1    ; Store the CSR and word count low.

      BICW     #<UCB$M_READ_READY!-
      UCB$M_WAITING_FOR_READ>,-
      UCB$W_DEVSTS(R5)              ; Clear the read ready flags.
      BICW     #<UCB$M_TIM!UCB$M_INT!UCB$M_TIMOUT!UCB$M_CANCEL!UCB$M_POWER>,-
      UCB$W_STS(R5)                  ; Clear unit status flags
      REQCOM                    ; Complete I/O in exec
      .DSABL LSB
      .PAGE

```

16

```
.SBTTL QK_INTERRUPT, Interrupt service routine for DR11-W
;
; ++
; QK_INTERRUPT, Handles interrupts generated by DR11-W
;
; Functional description:
;
; This routine is entered whenever an interrupt is generated
; by the DR11-W. It checks that an interrupt was expected.
; If not, it sets the unexpected (unsolicited) interrupt flag.
; All device registers are read and stored into the UCB.
; If an interrupt was expected, it calls the driver back at its Wait
; For Interrupt point.
; Deliver ATTN AST's if unexpected interrupt.
;
; Inputs:
;
; 00(SP) = Pointer to address of the device IDB
; 04(SP) = saved R0
; 08(SP) = saved R1
; 12(SP) = saved R2
; 16(SP) = saved R3
; 20(SP) = saved R4
; 24(SP) = saved R5
; 28(SP) = saved PSL
; 32(SP) = saved PC
;
; Outputs:
;
; The driver is called at its Wait For Interrupt point if an
; interrupt was expected.
; The current value of the DR11-W CSR's are stored in the UCB.
;
; --
QK_INTERRUPT:
    MOVL    @(SP)+,R4          ; Address of IDB and pop SP
    MOVQ    (R4),R4           ; CSR and UCB address from IDB

    DEVICELOCK -
        LOCKADDR=UCB$L_DLCK(R5),- ; Lock device access
        CONDITION=NOSETIPL,-      ; already at DIPL
        PRESERVE=NO               ; Don't preserve R0

; Check to see if device transfer request active or not
; If so, call driver back at Wait for Interrupt point and
; Clear unexpected interrupt flag.

    BBCC    #UCB$V_INT,UCB$W_STS(R5),24$
                                           ; If clear, no interrupt expected

; Read the DR11-W device registers (WCR, BAR, CSR) and store into UCB.

    BSBW    QK_REGISTER        ; Read device registers

    MOVL    UCB$L_FR3(R5),R3    ; Restore drivers R3
    JSB     @UCB$L_FPC(R5)      ; Call driver back
    BRB     25$

24$:  BSBW    QK_REGISTER        ; Read device registers
    INCW    UCB$W_QK_UNEXPECTED(R5) ; Increment Unexpected Interrupt count.

25$:  DEVICEUNLOCK -
        LOCKADDR=UCB$L_DLCK(R5),- ; Unlock device access
        PRESERVE=NO

    POPR    #^M<R0,R1,R2,R3,R4,R5> ; Restore registers
    REI                                ; Return from interrupt
```


VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```

.PAGE
17 .SBTTL QK_REGISTER - Handle DR11-W CSR transfers
;
; QK_REGISTER - Routine to handle DR11-W register transfers
;
; INPUTS:
;
; R4 - DR11-W CSR address
; R5 - UCB address of unit
;
; OUTPUTS:
;
; CSR, WCR, BAR, and status are read and stored into UCB.
; The DR11-W is placed in its initial state with interrupts enabled.
; R0 - .true. if no hard error
;       .false. if hard error (cannot clear ATTN)
;
; If the CSR ERROR bit is set and the associated condition can be cleared, then
; the error is transient and recoverable. The status returned is SS$_DRVERR.
; If the CSR ERROR bit is set and cannot be cleared by clearing the CSR, then
; this is a hard error and cannot be recovered. The returned status is
; SS$_CTRLERR.
;
; R0,R1 - destroyed, all other registers preserved.
;--

QK_REGISTER:

    MOVZWL QK_STATUS(R4),R1      ; Read STATUS.
    SWAPWORD R1
    MOVW   R1,UCB$W_QK_CSR(R5)   ; Save STATUS reg in UCB

    MOVW   #<QK_CONTROL$M_RPER!QK_CONTROL$M_RATN!QK_CONTROL$M_RDMA>,R0
    SWAPWORD R0
    MOVW   R0,QK_CONTROL(R4)     ; Clear all reset conditions in CSR.

    BBC    #QK_STATUS$V_ATTEN,R1,50$ ; Branch if not ATTN interrupt.
    BISW   #UCB$M_READ_READY,-      ; Indicate that the Read is Ready.
    UCB$W_DEVSTS(R5)

50$:  MOVZWL #SS$_NORMAL,R0        ; Assume success
    BBC    #QK_STATUS$V_PERR,R1,55$ ; Branch if no PARITY error
    MOVZWL #SS$_DRVERR,R0         ; Assume "drive" error
    BRB    60$

55$:  BBC    #QK_STATUS$V_BERR,R1,60$ ; Branch if no VMEbus error.
    MOVZWL #SS$_CTRLERR,R0        ; Assume "Controller" error.

60$:  MOVZWL QK_BAR_LOW_READ(R4),R1 ; Save the BAR LOW register in UCB.
    SWAPWORD R1
    MOVW   R1,UCB$W_QK_BARLOW(R5)

    MOVZWL QK_BAR_HIGH_READ(R4),R1 ; Save the BAR HIGH register in UCB.
    SWAPWORD R1
    MOVW   R1,UCB$W_QK_BARHIGH(R5)

    MOVZWL QK_WCR_LOW(R4),R1       ; Save the WCR LOW register in UCB
    SWAPWORD R1
    MOVW   R1,UCB$W_QK_WCRLOW(R5)

    MOVZWL QK_WCR_HIGH(R4),R1     ; Save the WCR HIGH register in UCB.
    SWAPWORD R1
    MOVW   R1,UCB$W_QK_WCRHIGH(R5)

    MOVW   #QK_CONTROL$M_IE,R1    ; Enable interrupts.
    SWAPWORD R1
    MOVW   R1,QK_CONTROL(R4)

```

```
100$:  MOVW    R0,UCB$W_QK_ERROR(R5)    ; Save error in UCB.
      RSB
```

18

```
      .SBTTL  QK_CANCEL, Cancel I/O routine
; ++
; QK_CANCEL, Cancels an I/O operation in progress
;
; Functional description:
;
;     Flushes Attention AST queue for the user.
;     If transfer in progress, do a device reset to DR11-W and finish the
;     request.
;     Clear interrupt expected flag.
;
; Inputs:
;
;     R2 = negated value of channel index
;     R3 = address of current IRP
;     R4 = address of the PCB requesting the cancel
;     R5 = address of the device's UCB
;
; Outputs:
;
; --
QK_CANCEL:                                ; Cancel I/O
      DEVICELOCK -
          LOCKADDR=UCB$L_DLCK(R5),- ; Lock device access
          SAVIPL=-(SP),-           ; Save current IPL
          PRESERVE=NO              ; Don't preserve R0
; Check to see if a data transfer request is in progress
; for this process on this channel
20$:  BBC     #UCB$V_INT,-           ; br if I/O not in progress
      UCB$W_STS(R5),30$
      JSB     G^IOC$CANCELIO        ; Check if transfer going
      BBC     #UCB$V_CANCEL,-
      UCB$W_STS(R5),30$             ; Branch if not for this guy
;
; Force timeout
;
      CLRL    UCB$L_DUETIM(R5)      ; clear timer
      BISW    #UCB$M_TIM,UCB$W_STS(R5) ; set timed
      BICW    #UCB$M_TIMEOUT,-
      UCB$W_STS(R5)                 ; Clear timed out
30$:
      DEVICEUNLOCK -
          LOCKADDR=UCB$L_DLCK(R5),- ; Unlock device access
          NEWIPL=(SP)+,-           ; Enable interrupts
          PRESERVE=NO
      RSB                                ; Return
```


VMS Version 5.4-3 Features

Sample Driver for a VMEbus DR11-W

```

19 .PAGE
.SBTTL QK_DEV_RESET - Device reset DR11-W
; ++
; QK_DEV_RESET - DR11-W Device reset routine
;
; This routine raises IPL to device IPL, performs a device reset to
; the required controller, and re-enables device interrupts.
;
; Must be called at or below device IPL to prevent a conflict in
; acquiring the device_spinlock.
;
; Inputs:
;
; R4 - Address of Control and Status Register
; R5 - Address of UCB
;
; Outputs:
;
; Controller is reset, controller interrupts are enabled
;
; --
QK_DEV_RESET:
    PUSHR    #^M<R0,R1,R2>          ; Save some registers
    DEVICELOCK -
        LOCKADDR=UCB$L_DLCK(R5),- ; Lock device access
        SAVIPL=-(SP),-           ; Save current IPL
        PRESERVE=NO              ; Don't preserve R0

    BSBB     QK_DEV_HWRESET

    DEVICEUNLOCK -
        LOCKADDR=UCB$L_DLCK(R5),- ; Unlock device access
        NEWIPL=(SP)+,-           ; Enable interrupts
        PRESERVE=NO

    POPR     #^M<R0,R1,R2>          ; Restore registers
    RSB

QK_DEV_HWRESET:
    MOVW     #QK_CONTROL$M_MCLR,R0  ; Issue a Master Clear to the device.
    SWAPWORD R0
    MOVW     R0,QK_CONTROL(R4)

; *** Must delay here depending on reset interval

    TIMEDWAIT TIME=#QK_RESET_DELAY ; No. of 10 micro-sec intervals to wait

    MOVW     #QK_CONTROL$M_IE,R0    ; Enable device interrupts
    SWAPWORD R0
    MOVW     R0,QK_CONTROL(R4)
    RSB

QK_END:                                     ; End of driver label
    .END

```


A.3.2 SCSI Device Support for the NCR 53C94 Controller

The VAXstation 4000 series systems now support one port (Port A) of an NCR 53C94 SCSI (Small Computer System Interface) controller module. The 53C94 controller supports both synchronous and asynchronous direct memory access (DMA) transfers that are controlled by the VAXstation 4000 I/O subsystem memory map registers. The existing SCSI data structures are described in the *VMS Device Support Reference Manual*.

The following sections describe the data structure changes needed to support the NCR 53C94 controller.

A.3.2.1 SCSI Device Driver Data Structures

Status bit 7 (SPDT\$V_FIFOLCK) is added to longword SPDT\$L_STS in the SCSI Port Descriptor Table (SPDT). When the FIFO buffer is in use, the port driver sets SPDT\$V_FIFOLCK.

Longword SPDT\$L_PORT_FLAGS now contains the byte count divisor in bits <31:25>. This provides the class driver with a *suggested* data transfer byte count for the port.

Flag bit 4 (SCDRP\$V_MREG_DONE) is added to longword SCDRP\$L SCSI_FLAGS in the SCSI Class Driver Request Packet (SCDRP) descriptor. The port driver sets SCDRP\$V_MREG_DONE when the map registers are loaded to control the data transfer.

A.3.2.2 Using the SPI\$CONNECT Macro and Maximum Byte Counts

As described in the *VMS Device Support Manual*, the SCSI port supports a maximum data transfer byte count value. The port driver returns this value (SPDT\$L_MAXBYTECNT) in R1 when the class driver invokes the SPI\$CONNECT macro. Some devices, typically tape drives, need to use the full value of SPDT\$L_MAXBYTECNT. Most devices, such as disk drives, can better utilize resources with a smaller (suggested) byte count for each DMA transfer. The class driver can derive the suggested byte count by utilizing a divisor value in bits <31:25> of the port capability mask (SPDT\$L_PORT_FLAGS longword) returned by SPI\$CONNECT in R3. For example, if the maximum byte count is 64K and the divisor is 4, then the class driver calculates the suggested byte count as 16K. A sample code sequence (that follows the execution of SPI\$CONNECT) for the class driver to calculate the suggested byte count is:

```

;*****
; After SPI$CONNECT execution, R3 contains divisor value in
; <31:25> and R1 contains MAXBYTECNT
ASHL    #-24,R3,R3      ;Shift divisor value to low-order byte of R3
DIVL3   R3,R1,R0        ;Divide MAXBYTECNT (R1) by divisor (R3) and
                        ;and place suggested byte count in R0

```

A.3.3 FDDI and Ethernet—VMS Support

This section details changes and additions made to the VMS programming interface for fiber distributed data interface (FDDI) and Ethernet, described in the *VMS I/O User's Reference Manual: Part II*. This section includes a brief discussion of FDDI, a comparison between FDDI and Ethernet, and some background on the changes and additions that have been made to the programming interface. A complete reference of the various frame and packet formats VMS supports is also included.

A.3.3.1 Overview of FDDI

FDDI is Digital's next generation of the local area network (LAN). FDDI has advantages over Ethernet that include 10 times the data rate and significant increases in LAN diameter. The first VMS device or network adapter for FDDI is the DEC FDDIcontroller 400 (DEMFA) for VAX systems based on XMI (6000 and 9000 class).

The one change required to an existing VMS Ethernet application is that it must be able to find the new device (FXA0:). Other changes can be made to VMS Ethernet applications to take advantage of the larger frame and message size that FDDI can carry.

A.3.3.2 New FDDI Device DEMFA

The DEMFA device and FDDI are supported by both DECnet-VAX Phase IV and DECnet-VAX Extensions. The new NCP line/circuit name takes the form MFA-x (x=0,1,...). FDDI/DEMFA is also supported by the LAT software. You can specify the new FDDI device name (FXA0:) when defining the LAT\$LINK; see the *VMS LAT Control Program (LATCP) Manual* for more details.

The DEMFA and FXDRIVER can support only one multicast user at a time. Also, the maximum number of channels that can be assigned to a single DEMFA device is 29 (DECnet, LAT, and clusters each count as one channel). Digital may change or remove these restrictions on DEMFA support in the future.

A.3.3.3 Programming Interface

Existing Ethernet applications must be able to locate the new FDDI device (FXcu:). Applications that currently translate logical names to locate the valid Ethernet device can operate on FDDI. All that needs to change is the logical name definition, which must have FXcu defined as a valid translation.

VMS currently supports the following LAN devices (CMSA/CD stands for carrier sense, multiple access with collision detect):

VMS Device Name	Device	LAN Type
FXcn:	DEMFA	FDDI
XEcn:	DELUA/DEUNA	CSMA/CD
XQcn:	DEQNA/DEQTA/DELQA	CSMA/CD
ETcn:	DEBNA/DEBNI	CSMA/CD
EXcn:	DEMNA	CSMA/CD
EZcn:	SGEC	CSMA/CD
EScn:	LANCE	CSMA/CD

Note

The *c* in the VMS device name indicates the controller letter. The *n* in the VMS device name indicates the unit number.

Even though FDDI is a new type of LAN, there are no new packet formats defined for the programming interface. The only packet formats supported by VMS for FDDI are the three existing packet formats NMA\$C_LINFM_ETH, NMA\$C_LINFM_802, and NMA\$C_LINFM_802E (refer to Section A.3.3.5).

At the option of the application programmer, changes can be made to take advantage of the larger frame and message size of FDDI. To do that, the application cannot treat FDDI and Ethernet identically. The application must use new and additional features of the programming interface to determine and monitor the capabilities of the device, the LAN, and the remote device and application's ability to communicate using large frames.

An application can determine if a device is an FDDI device if an NMA\$C_PCLI_MED request returns a value of NMA\$C_LINMD_FDDI; see Section A.3.3.4.1.

The application can determine and monitor the capability of the LAN and the remote computer and application to exchange the larger FDDI frames by examining the priority bits of the Frame Control (FC) field for FDDI frames received. For more information about the Frame Control field, see Section A.3.3.5. By specifying the new NMA\$C_PCLI_RFC parameter, applications can receive a copy of the FC (one byte) field of the FDDI frames received. VMS puts the received FC in the P5/P6 buffer associated with the \$QIO Read operation. The required size of the P5/P6 buffer is increased by one when NMA\$C_PCLI_RFC is specified. The priority bits are the three low-order bits of the FC. If these bits are zero, it indicates that small buffers must be used. If the value is nonzero, larger messages may be exchanged. A brief discussion of this property of the priority bits of the FC is included in Section A.3.3.5. The application must also post larger buffers for the receive operation and specify a larger value for NMA\$C_PCLI_BUS. It is necessary to continually monitor the value of the priority bits in the FC of received frames in order to properly communicate using large buffers. Failure to do so can result in a change in network topology, such that two computers that were connected by only FDDI links now have an Ethernet segment in the path. The application gets no indication that small buffers must be used, except for the changes in the value of the received FC priority bits.

VMS Version 5.4-3 Features Programming Interface

By default, the VMS device drivers on transmit insert zero into the priority field of the FC. In this way, Ethernet applications that have not yet been modified send the correct FC to indicate that they do not support large frames. For applications that want to support large frames, NMA\$C_PCLI_XFC must be specified to get a nonzero value for the priority bits in the transmit FC. This is so that remote nodes and applications will see a nonzero value in the priority bits of the FC they receive remotely, indicating support for large frames at the local node and application.

A.3.3.4 Parameters

Effective with VMS Version 5.4-3, the P2 extended characteristics buffer parameters NMA\$C_PCLI_BSZ and NMA\$C_PCLI_DCH are obsolete. Both the application and VMS ignore these parameters if they are specified.

Descriptions of new and changed parameters for FDDI follow.

A.3.3.4.1 NMA\$C_PCLI_MED (Medium) This read-only parameter is new. It is a byte-type parameter and returns the type of medium to which the device is attached to. Current media supported are NMA\$C_LINMD_FDDI (FDDI) and NMA\$C_LINMD_CSMACD (Ethernet and IEEE 802.3).

A.3.3.4.2 NMA\$C_PCLI_RFC (Receive Frame Control) NMA\$C_PCLI_RFC is a new optional parameter for FDDI devices only (specifying NMA\$C_PCLI_RFC for an Ethernet device causes the application to receive an SS\$_BADPARAM error). It is a byte type with the following values:

- NMA\$C_STATE_ON—Application gets a copy of the FC for each FDDI frame received.
- NMA\$C_STATE_OFF—Application does not get a copy of the FC for received FDDI frames (this is the default).

For \$QIO Read operations, the FC is passed to the application in the P5 diagnostics buffer. Table A-6 lists the size required for the P5 diagnostics buffer for various packet formats and settings of NMA\$C_PCLI_RFC.

Table A-6 Required Size for P5 Diagnostics Buffer on FDDI Devices

Packet Format	NMA\$C_STATE_OFF	NMA\$C_PCLI_ON
Ethernet (NMA\$C_LINFM_ETH)	14	15
802 (NMA\$C_LINFM_802E)	16	17
802E (NMA\$C_LINKFM_802E)	20	21

Receiving the FC requires one additional byte of space in the P5 buffer. The FC is the first byte in the P5 buffer, immediately preceding the 6-byte Destination Address. You may notice that the size of the P5 buffer required does not change from Ethernet if NMA\$C_PCLI_RFC is NMA\$C_STATE_OFF.

A.3.3.4.3 NMA\$C_PCLI_XFC (Transmit Frame Control) NMA\$C_PCLI_XFC enables applications to control the setting of the priority bits in the FC for frames being transmitted in a \$QIO Write operation. It is a byte parameter that has many valid settings. If specified with a value of zero, the application supplies an FC value on each \$QIO Write operation. The FC value to be used in this case is supplied in the P5 buffer for the \$QIO Write operation. If the parameter is specified with a nonzero value, then that value is inserted into the FC field of every transmission by the VMS device drivers. No FC is present in the P5 buffer for the \$QIO Write operation in this case. If this parameter is not specified, the default setting (zero) of the priority bits is used.

Regardless of how the FC is supplied, the value specified must be valid. The allowable values for FC are between FC_LLC_MIN and FC_LLC_MAX. If NMA\$C_PCLI_XFC is specified with a nonzero value outside the valid range, the application receives a SS\$_BADPARAM error. FC_LLC_MIN is 80 (50 hex), and because the priority bits are the three low-order bits, FC_LLC_MAX is 87 (57 hex). An application that needs to support large buffers should use a value for NMA\$C_PCLI_XFC that is at least one greater than FC_LLC_MIN, because the priority bits are zero (50 hex has all three low-order bits zeroed) and this would not indicate correctly that large buffers are supported.

A.3.3.4.4 NMA\$C_PCLI_BUS (Maximum Receive Buffer Size) NMA\$C_PCLI_BUS is a word-type parameter that enables applications to specify the maximum size of their receive buffers. This parameter can now have a maximum value of 4478, reflecting the maximum data size that can be carried by an FDDI frame. If a value greater than 4478 is specified, the application receives no frames nor any indication of loss of frames that could not be delivered because they were too large.

A.3.3.4.5 NMA\$C_PCLI_MBS (Maximum Packet Length) NMA\$C_PCLI_MBS is a word-type, read-only parameter. The value returned reflects the largest data packet that the application can receive for its packet format and type of LAN, measured in bytes. The values are:

Packet Format	CSMA/CD	FDDI
Ethernet	1500	4470
Ethernet w/PAD	1498	4468
802	1497	4475
802E	1492	4470

A.3.3.4.6 NMA\$C_PCLI_CCA (Can Change Address) This new parameter enables applications that want to start before DECnet on a VMS system, but do not know the DECnet address, to start without affecting the ability of DECnet to start. DECnet attempts to set the physical address of the Ethernet controller when it starts. Ethernet devices support only one physical address, so all applications that are using the same Ethernet device must also use the same physical address. If another application that does not use the DECnet address starts before DECnet, DECnet will not be able to start on that Ethernet controller unless other applications that have already started have all specified NMA\$C_PCLI_CCA.

VMS Version 5.4-3 Features Parameters

This parameter is not applicable to FDDI, because FDDI devices can run with more than one physical address. However, no error is returned if this parameter is supplied for FDDI devices.

The application receives no indication that the physical address has changed.

A.3.3.5 Frame and Packet Formats

This section illustrates the various frame and packet formats that are supported by VMS.

A.3.3.5.1 FDDI Frames Figure A-7 illustrates the format of FDDI frames.

Figure A-7 FDDI Frame Format

FC	DA	SA	DATA	CRC
1	6	6	0=>4478	4

- Minimum length -- 17 bytes (no data)
- Maximum length -- 4495 bytes

FC: Frame Control contains a "priority" field which can be used to determine if the frame originated on the FDDI, or on the Ethernet.

DA: Destination Address

SA: Source Address

CRC: Cyclic Redundancy Check

ZK-3742A-GE

A.3.3.5.2 CSMA/CD Frames

There are three formats for CSMA/CD frames:

- Ethernet format
- Ethernet format with PAD option
- IEEE 802.3 format

Figure A-8 illustrates the Ethernet format.

Figure A-8 Ethernet Frame Format

DA	SA	PTY	DATA	CRC
6	6	2	46=>1500	4

Minimum total length – 64 bytes

Maximum total length – 1518 bytes

DA: Destination Address

SA: Source Address

PTY: Ethernet Protocol Type

DATA: User's data (can include 2-byte length field)

CRC: Cyclic Redundancy Check

ZK-3743A-GE

Ethernet frames must be a minimum of 64 bytes in length, which means that the minimum data length is 46 bytes. Applications select Ethernet format by specifying NMA\$C_LINFM_ETH as the value for NMA\$C_PCLI_FMT in their P2 characteristics buffer.

If the amount of data to be transmitted is less than 46 bytes, the VMS CSMA/CD drivers transmit extra bytes of zero after the application data. VMS has provided the Ethernet format with PAD option to alleviate this problem. Figure A-9 illustrates the Ethernet format with the PAD option.

Figure A-9 Ethernet Frame Format with PAD Option

DA	SA	PTY	PAD	DATA	CRC
6	6	2	2	46=>1500	4

PAD: The actual length of the requested transmission data

ZK-3744A-GE

The Ethernet with PAD format differs from the standard Ethernet format because two bytes from the data portion are used to provide a PAD field that contains the length of the transferred data. Extra zeros added to meet the 64-byte minimum length requirement are not included in the length of the received data reported to an application using the PAD option. The VMS LAN drivers (CSMA/CD and FDDI) insert and remove the PAD values transparently. A receiving application

VMS Version 5.4-3 Features

Frame and Packet Formats

does not see the PAD field in its data buffer and a transmitting application need not specify the PAD value because the VMS LAN drivers manage the PAD field.

Applications select the PAD option by adding the optional NMA\$C_PCLI_PAD with a value of NMA\$C_STATE_ON in their P2 characteristics buffer, along with selecting NMA\$C_LINFM_ETH for the NMA\$C_PCLI_FMT parameter.

Figure A-10 illustrates the IEEE 802.3 format.

Figure A-10 IEEE 802.3 Frame Format

DA	SA	LEN	DATA	CRC
6	6	2	46=>1500	4

DA: Destination Address

SA: Source Address

LEN: The length of data portion only. It can be less than 46 if user supplied small data, but the frame is then padded to meet minimum length requirements.

DATA: Users data (may include z-byte length field)

CRC: Cyclic Redundancy Check

ZK-3745A-GE

The IEEE 802.3 format is similar to the Ethernet format, except the PTY field is replaced by the LEN field. The LEN field performs the same function as the PAD field in the Ethernet with PAD format.

A.3.3.5.3 Packet Formats The Ethernet packet format for CSMA/CD frames is identical to the Ethernet frame format shown in Figure A-8. The data portion of the Ethernet packet is the application data.

Figure A-11 shows the mapped Ethernet packet format for FDDI Frames.

Figure A-11 FDDI Frame with Mapped Ethernet Packet Format

FC	DA	SA	SNAP SAP	SNAP PID	PTY	DATA	FCS
1	6	6	3	3	2	X	4

SNAP SAP = AA-AA-03 (hex)

SNAP PID = 00-00-00

PTY = Ethernet PTY

DATA length 0 <= X <= 4470

If PAD 0 <= X <= 4468

ZK-3746A-GE

This is the structure of an FDDI frame that is produced by applications using Ethernet packet formatting (NMA\$C_LINFM_ETH) on the FDDI.

Figure A-12 shows the mapped Ethernet with PAD option packet format for FDDI frames.

Figure A-12 FDDI Frame with Mapped Ethernet with PAD Option Packet Format

FC	DA	SA	SNAP SAP	SNAP PID	PTY	PAD	DATA	FCS
1	6	6	3	3	2	2	X	4

SNAP SAP = AA-AA-03 (hex)

SNAP PID = 00-00-00

PTY = Ethernet PTY

DATA length 0 <= X <= 4470

ZK-3747A-GE

This is the structure of an FDDI frame that is produced by applications using Ethernet packet formatting with the PAD option on VMS FDDI devices. The meaning and function of the PAD field are the same for both FDDI and Ethernet (see Figure A-9). As is true with Ethernet, there is a corresponding 2-byte decrease in the maximum data length when using the PAD option with FDDI.

Figure A-13 shows an FDDI frame with 802 packet format.

Figure A-13 FDDI Frame with 802 Packet Format

FC	DA	SA	DSAP	SSAP	CTL	DATA	FCS
1	6	6	1	1	1-2	Z	4

DATA length 0 <= Z <= 4475 (1 byte CTL)

<= Z <= 4474 (2 byte CTL)

ZK-3748A-GE

Figure A-14 shows a CSMA/CD frame with 802 packet format.

Figure A-14 CSMA/CD Frame with 802 Packet Format

DA	SA	LEN	DSAP	SSAP	CTL	DATA	CRC
6	6	2	1	1	1-2	N	4

DSAP = Destination Service Access Point

SSAP = Source Service Access Point

CTL = Control Field

DATA length 0 <= N <= 1497 (1 byte CTL)

<= N <= 1496 (2 byte CTL)

ZK-3749A-GE

VMS Version 5.4-3 Features

Frame and Packet Formats

Figure A-15 shows an FDDI frame with 802E packet format.

Figure A-15 FDDI Frame with 802E Packet Format

FC	DA	SA	SNAP SAP	PID	DATA	FCS
1	6	6	3	5	Y	4

PID = User's Protocol Identifier

DATA length $0 \leq Y \leq 4470$

ZK-3750A-GE

Figure A-16 shows the 802E, 802.1 SNAP, and 802.1 PID packet formats for CSMA/CD frames.

Figure A-16 CSMA/CD Frames, 802E, 802.1 SNAP, and 802.1 PID Packet Format

DA	SA	LEN	SNAP SAP	PID	DATA	CRC
6	6	2	3	5	M	4

PID = User's Protocol Identifier

DATA length $0 \leq M \leq 1492$

ZK-3751A-GE

A.3.4 Preferred Access Path Programming Examples

In VMS Version 5.4, a new disk class driver QIO function was added to enable a user program to specify a preferred access path (IO\$_SETPRFPATH). This feature is described in the *VMS Version 5.4 New Features Manual* and the *VMS I/O User's Reference Manual: Part I*.

In response to customer requests, two sample programs for this feature have been added to the VMS Version 5.4-3 software distribution kit. Written in BLISS-32 and VAX MACRO languages, they can be found in SYS\$EXAMPLES:PREFER.*.

A.3.5 VAX Ada Run-Time Library

The following new features have been added to the VMS Version 5.4-3 VAX Ada Run-Time Library (ADARTL.EXE):

- The precision of delay statements has been improved. Previously, one clock tick was added to all delay values; this no longer occurs.
- The implementation of the DELETE procedures provided by the Ada input-output packages has changed. There are now four possible results when

you attempt to delete a file (prior to this release, only the first three results occurred):

- The DELETE procedure succeeds: the file is closed and deleted.
- The file was already closed; the exception STATUS_ERROR is raised.
- An error occurs: an exception such as USE_ERROR is raised, but the file is left open.
- An error occurs: an exception such as USE_ERROR is raised; the file is closed but not deleted.

In the cases where an error occurs, you can determine if the file has been left open or closed by first handling the exception and then calling the Ada input-output function IS_OPEN.

- The exception USE_ERROR is no longer raised for correct FORM parameter values in the procedure INDEXED_IO.OPEN.
- Certain VFC formatted text files are treated as files with lines of indefinite length.

Starting with VAX Ada Version 2.2, the implementation of the package TEXT_IO supports use of the form string to create external text files with lines of indefinite length. (Previously, the length of text-file lines was determined by the maximum length of a VMS RMS record.)

VAX Ada recognizes files with the following characteristics as files of indefinite line length:

- The print form of carriage control
- A 2-byte header size (will apply to all records in the external file)
- Variable-length with fixed-length control field (VFC) record format
- A maximum record size of zero

Lines are written to files with indefinite line length as one or more VMS RMS records. The characters in the 2-byte header of each record keep track of which records comprise the beginning, middle, and end of a line. For more information about files with indefinite line length, see the release notes for VAX Ada V2.2.

Note that in some cases you may want to open a text file that has the characteristics of an indefinite-line-length file (for example, a file created by some other VMS-related software). If you do not want the file to be treated as one with indefinite line length, then open the file with the TEXT_IO.OPEN procedure and specify a nonzero record length in the form string. For example:

```
TEXT_IO.OPEN(  
  FILE => FIXED_LINE_LENGTH_FILE,  
  FORM => "RECORD;"  
           "SIZE 1;";)
```

Regardless of its value, the only effect of the nonzero record length in this case is to prevent the file from being treated as one with indefinite line length.

A.3.6 DECwindows X11 Display Server—Color Name File

The file DECW\$RGB.COM now contains all of the color names and values that were distributed with MIT X Window System Version 11 Release 4 (MIT X11 Release 4). The colors are listed in DECW\$RGB.COM in alphabetical order so that they can be easily compared with the table of common color names published in the *X and Motif Quick Reference Guide*.

Following the list of MIT X11 Release 4 common color names in the file DECW\$RGB.COM is a list of a set of the Digital-specific color names and their values. These names include DECWBlue and a set of color names for the default DECwindows system colors.

The color value for DECWBlue has been changed to better reflect the Digital corporate standard for Digital Blue.

The following color names are the default DECwindows system colors:

- Screen background
- Border topshadow
- Border background
- Border bottomshadow
- Window topshadow
- Window background
- Window bottomshadow

These color names are also specified without spaces in the names (for example, ScreenBackground), as are all of the MIT X11 Release 4 common color names.

Following the Digital-specific color names in DECW\$RGB.COM are four additional color names and values that were included in previous releases of the DECwindows systems but that are not part of the MIT X11 Release 4 colors. These colors are:

- Medium forest green
- MediumForestGreen
- Medium goldenrod
- MediumGoldenrod

A.3.7 Changes to SDA SHOW PORTS Command

VMS Version 5.4-3 includes changes to the System Dump Analyzer (SDA) command SHOW PORTS that allow you to view the data structures that the multiadapter local area cluster uses. For more information about multiadapter local area clusters, see the *VMS VAXcluster Manual*.

In VMS Version 5.4-3, the SHOW PORTS command has the following additional qualifiers:

- /BUS[=bus-address]: Displays BUS (LAN device) structure data
- /CHANNEL[=channel-address]: Displays channel (CH) data

VMS Version 5.4-3 Features Changes to SDA SHOW PORTS Command

- /DEVICE: Displays the network path description for a channel
- /MESSAGE: Displays the message data associated with a virtual circuit (VC)
- /VC[=vc-address]: Displays the virtual circuit (VC) data

In VMS Version 5.4-3, the SHOW PORTS command also defines symbols based on the cluster configuration. These symbols include the following information:

- Virtual circuit (VC) control blocks for each of the remote systems
- BUS data structure for each of the local LAN adapters
- Some of the data structures used by both PEDRIVER and the LAN driver

The following symbols are defined automatically:

- VC_nodename: Example: VC_NODE1, address of the local node's virtual circuit to node NODE1.
- CH_nodename: The preferred channel for the virtual circuit. For example, CH_NODE1, address of the local node's preferred channel to node NODE1.
- BUS_busname: Example: BUS_ETA, address of the local node's BUS structure associated with BUS ETA.
- PE_PDT: Address of PEDRIVER's port descriptor table.
- MGMT_VCRP_busname: Example: MGMT_VCRP_ETA, address of the management VCRP for BUS ETA.
- HELLO_VCRP_busname: Example: HELLO_VCRP_ETA, address of the HELLO message VCRP for BUS ETA.
- VCIB_busname: Example: VCIB_ETA, address of the VCIB for BUS ETA.
- UCB_LAVC_busname: Example: UCB_LAVC_ETA, address of the LAN device's UCB used for the local area VAXcluster protocol.
- UCB0_LAVC_busname: Example: UCB0_LAVC_ETA, address of the LAN device's template UCB.
- LDC_LAVC_busname: Example: LDC_LAVC_ETA, address of the LDC structure associated with LAN device ETA.
- LSB_LAVC_busname: Example: LSB_LAVC_ETA, address of the LSB structure associated with LAN device ETA.

These symbols equate to system addresses for the corresponding data structures. You can use these symbols, or an address, after the equal sign (=) in the following commands:

- SHOW PORTS /BUS=bus-address: Displays the data for the specified BUS structure. The last event time is at the top of the lower right-hand column. If an error was counted, the last error time is displayed under Xmt Errors. The normal status is RUN, ONLINE, and RESTART.

VMS Version 5.4-3 Features

Changes to SDA SHOW PORTS Command

SDA> SHOW PORTS /BUS=BUS_ESA
VAXcluster data structures

```

-----
--- BUS: 80B08090 (ESA) Device: ES_LANCE LAN Address: AA-00-04-00-33-FD---
                        LAN Hardware Address: 08-00-2B-12-AE-A1
Status: 00000A03 run,online,xmt_chaining_disabled,restart
----- Transmit ----- Receive ----- Structure Addresses ---
Msg Xmt      434107  Msg Rcv      1170090  PORT Address      80B091B
Mcast Msgs   103939  Mcast Msgs   859601  VCIB Addr         80B08248
Mcast Bytes  13304192 Mcast Bytes  96272072 HELLO Message Addr 80B082D8
Bytes Xmt     59789962 Bytes Rcv    146674695 BYE Message Addr   80B08468
Outstand I/Os 0      Buffer Size    1424  Delete BUS Rtn Adr 8079E424
Xmt Errors    75     Rcv Ring Size    8
Last Xmt Error 00000334 Time of Last Xmt Error 25-MAR-1991 23:39:28.27
--- Receive Errors --- BUS Timer ----- Datalink Events -----
TR Mcast Rcv 0      Handshake TMO 8079FA50 Last 22-MAR-1991 18:25:25.12
Rcv Bad SCSID 0      Listen TMO 8079FA54 Last Event      00001202
Rcv Short Msg 0      HELLO timer 1      Port Usable      1
Fail CH Alloc 0      HELLO Xmt err 38  Port Unusable     0
Fail VC Alloc 0      Address Change 1
Wrong PORT 0      Port Restart Fail 0

```

- **SHOW PORTS /VC=vc-address:** Displays the virtual circuit data for the specified remote node and a channel summary. In the following example, the upper center of the the display contains the virtual circuit status. The lower right-hand corner contains the virtual circuit open and close times.

SDA> SHOW PORTS/VC=VC_BREE
VAXcluster data structures

```

-----
--- Virtual Circuit (VC) 806CD6E0 ---
Remote System Name: BREE (0:VAX) Remote SCSSYSTEMID: 64856
Local System ID: 222 (DE) Status: 0005 open,path
----- Transmit ----- VC Closures ----- Congestion Control -----
Msg Xmt      216686  SeqMsg TMO 0      UnAcked Msgs      1
Unsequence    3      CC DFQ Empty 0      Pipe Quota Reached 33
Sequence     149643  Topology Change 0      CMD Queue Len      0
ReXmt         545   NPAGEDYN Low 0      Max CMD Queue Len   5
Lone ACK       66495  RSVP Threshold 15
Bytes Xmt     33309074 Pipe Quota 31
----- Receive ----- :Messages Discarded: ----- Channel Selection -----
Msg Rcv      194492  No Xmt Chan 0      Preferred Channel 80704320
Unsequence    1      Rcv Short Msg 0      Delay Time      FB7E6F80
Sequence     178905  Illegal Seq Msg 0      Buffer Size     1424
ReRcv        30      Bad Checksum 0      Channel Count    6
Lone ACK     15531  TR DFQ Empty 0      Channel Selections 3920
Cache        26      TR MFQ Empty 0      Protocol        1.3.0
Ill ACK       0      CC MFQ Empty 0      Open 1-JAN-1991 00:00:07.03
Bytes Rcv    52086897 Cache Miss 0      Cls 17-NOV-1858 00:00:00.00

```

Press RETURN for more.
VAXcluster data structures

: Channel Summary for Virtual Circuit (BREE) 806CD6E0 --

Address	Type	Xmt Time	Size	Preferred	Best	Last State Change
80704320	Preferred	FB7E6F80	1424	812	617	22-MAR-1991 18:14:07.01
807043E0	Active	FB7E735E	1424	95	4	25-MAR-1991 20:01:15.18
807050D0	Active	FB7E7FED	1424	431	0	25-MAR-1991 20:01:15.18
806CD820	Active	FB7E728E	1424	868	1470	25-MAR-1991 20:01:15.18
80705010	Active	FB7E7043	1424	738	9	25-MAR-1991 20:00:58.17
806CD8E0	Active	FB7E7BB5	1424	976	1744	25-MAR-1991 20:00:31.17

VMS Version 5.4-3 Features Changes to SDA SHOW PORTS Command

- **SHOW PORTS /CHANNEL=channel-address:** Displays the data for the specified channel. The normal state is OPEN, with a status of PATH, OPEN, and RMT_HWA_VALID.

In the following example, the top of the display shows the remote device name, remote device type, and the channel open and close times.

```
SDA> SHOW PORTS/CHANNEL=CH_BREE
VAXcluster data structures
-----
: PEDRIVER Channel (CH:80704320) for Virtual Circuit (VC:806CD6E0) BREE  --
State: 0004 open                      Status: 0B path,open,rmt_hwa_valid
BUS: 80B008B0 (XQA) Lcl Device: XQ_DELQA Lcl LAN Address: 08-00-2B-0A-6A-6B
Rmt Name: XQB                      Rmt Device: XQ_DEQTA Rmt LAN Address: 08-00-2B-13-70-88
Rmt Seq #: 0002 Open:22-MAR-1991 18:14:07.01 Closed:17-NOV-1858 00:00:00.00
----- Transmit ----- Receive ----- Channel Selection -----
Lcl CH Seq #      0001 Msg Rcv      139205 Average Xmt Time FB879740
Msg Xmt           66707 Mcast Msgs   103906 Remote Buffer Size   1424
Ctrl Msgs         1 Mcast Bytes 10182788 Max Buffer Size     1424
Ctrl Bytes        98 Ctrl Msgs      2 Best Channel         615
Bytes Xmt        9130385 Ctrl Bytes   196 Preferred Channel    810
Rmt Ring Size     31 Bytes Rcv      22654333 Retransmit Penalty    2
----- Channel Errors ----- Xmt Error Penalty    12
Handshake TMO     0 Short CC Msgs    0 ----- Channel Timer -----
Listen TMO        0 Incompat Chan    0 Timer Entry Flink 8079FA3C
Bad Authorize     0 No MSCP Srvr    0 Blink 80705010
Bad ECO           0 Disk Not Srvd    0 Last Ring Index    08
Bad Multicast     0 Old TR Msgs    0 Protocol          1.3.0
Topology Change   0 Supported Services 00000000
```

- **SHOW PORTS /CHANNEL /VC=vc-address:** Displays the following information:
 - Virtual circuit data for the specified remote node
 - Channel data associated with each of the channels to the remote node
- **SHOW PORTS /DEVICE /CHANNEL /VC=vc-address:** Displays the following information:
 - Virtual circuit data for the specified remote node
 - Channel data and network path description for each channel to the remote node

```
VAXcluster data structures
-----
: Network Component List (CLST:80D36440) for Channel (CH:806DC420) --
COMP adr  COMP Type  Description
-----
80D30010  NODE        SGRPOP:VAXstation 3300; ZK03-4/U10
80CC9300  ADAPTER     ESA; SGRPOP:VAXstation 3300; ZK03-4/U10 (08-00-2B-12-AE-A1)
80D3CDB0  COMPONENT   ZK34C4, I-Cluster Segment DEMPR
80D40380  COMPONENT   ZK34C4, I-Cluster Segment DELNI
80D36AD0  COMPONENT   I-Cluster Segment
80D2D4C0 P COMPONENT   ZK03-4 Lab, LIVER: I-Cluster Segment DELNI
80CC2BE0 S ADAPTER     XQA; DELLNM:rack mounted MicroVAX II; ZK03-4 Lab (08-00-2B-0C-C4-1D)
80D323F0  NODE        DELLNM:rack mounted MicroVAX II; ZK03-4 Lab
```

This display is useful after the local area VAXcluster network failure analysis data has been loaded. After a network failure analysis, this display indicates primary and secondary failed component suspects in the following ways:

- P: Primary suspect
- S: Secondary suspect

VMS Version 5.4-3 Features

Changes to SDA SHOW PORTS Command

- ?: Component that cannot be proved to be working
 - SHOW PORTS /MESSAGE /VC=vc-address: Displays the virtual circuit data for the specified remote node, followed by the message data for the remote node. The virtual circuit message display shows the counters for the following items:
 - Sequenced message delivery
 - Any messages in the process of being transmitted or in the receive cache
- The following is an example of a display resulting from the SHOW PORTS /MESSAGE /VC=vc-address command:

VAXcluster data structures

```
-----
--- Sequenced Message Counters Virtual Circuit (VC) 806CD6E0 ---
NSU: 4457   HAA: 4456   LAR: 4455   HSR: B3AA   Cache Mask: 00000000

      Messages Waiting for ACKs
VCRP adr  Len  Flgs Seq  Ack  Message Data
-----
806CD2E0   137  0B  4456  B3AA  02 7D 00 04 00 0A 00 00 00 09 00 D 75 05 00 67
```

- SHOW PORTS /ADDRESS=PE_PDT: Displays the following information:
 - Port descriptor table (PDT) structure
 - Some of the fields in the port structure
 - BUS summary
 - Virtual circuit summary

The following is an example of a display resulting from the SHOW PORTS /ADDRESS=PE_PDT command:

SDA> SHOW PORTS /ADDRESS=PE_PDT
VAXcluster data structures

```
-----
--- Port Descriptor Table (PDT) 806C37A0 ---

Type: 03 pe
Characteristics: 0000

Msg Header Size      32  Connect      80799F94  Recyclh_Msg_Buf  8079AD8A
Max Xfer Bcnt        FFFFFFFF Dealloc_Dg_Buf  8079AFDA  Request_Data     8079B1CC
DG Header Size       288  Disconnect   8079A06B  Send_Data        8079B215
Poller Sweep         31  Unmap        8079B510  Send_Dg_Buf      8079B03E
Fork Block W.Q.      empty Map          8079B111  Send_Msg_Buf     8079AEA8
UCB Address          806C0E50 Map_Bypass    8079B0F8  Send_Cnt_Msg_Buf 8079AEAF
ADP Address          00000000 Map_Irp       8079B101  Read_Count       80796D59
Accept              80799FEC Map_Irp_Bypass 8079B0F0  Rls_Read_Count   80796DD3
Alloc_Dg_Buf        8079AFC6 Queue_Dg_Buf  8079AFE0  Mreset           80799C94
Alloc_Msg_Buf       8079AD05 Queue_Mult_Dgs 8079AFE8  Mstart           80799C9E
Dealloc_Msg_Buf     8079ADE3 Recycl_Msg_Buf 8079AD94  Stop_Vcs         8079BEDD
Dealloc_Msg_Buf_Reg 8079ADF6 Reject        8079A036  Send_Dg_Reg      8079B031
```

Press RETURN for more.
VAXcluster data structures

```
-----
--- Port Block 80B091B0 ---

Status: 0001 authorize
VC Count: 5
Secs Since Last Zeroed: 311728
```

VMS Version 5.4-3 Features Changes to SDA SHOW PORTS Command

SBUF Size	436	LBUF Size	1788
SBUF Count	12	LBUF Count	1
SBUF Max	768	LBUF Max	384
SBUF Quo	13	LBUF Quo	1
SBUF Miss	18	LBUF Miss	12235
SBUF Allocs	499579	LBUF Allocs	16824
SBUFs In Use	0	LBUFs In Use	0
Peak SBUF In Use	14	Peak LBUF In Use	34
SBUF Queue Empty	0	LBUF Queue Empty	0
TR SBUF Queue Empty	0		
No SBUF for ACK	0		

Bus Addr	Bus	LAN Address	Error Count	Last Error	Time of Last Error
80B08920	LCL	00-00-00-00-00-00	0		
80B08090	ESA	AA-00-04-00-33-FD	75	00000334	25-MAR-1991 23:39:28.27
80B008B0	XQA	08-00-2B-0A-6A-6B	12	0000002C	23-MAR-1991 12:43:59.07
80AF6E90	XQB	08-00-2B-08-CB-B8	0		

Press RETURN for more.
VAXcluster data structures

--- Virtual Circuit (VC) Summary ---

VC Addr	Node	SCS ID	Lcl ID	Status Summary	Last Event Time
806CD1A0	NODE12	64819	223/DF	open,path	1-JAN-1991 00:00:00.03
806CD6E0	NODE13	64856	222/DE	open,path	1-JAN-1991 00:00:07.
806CD9A0	NODE14	64587	221/DD	open,path	22-MAR-1991 18:34:10.18
8070D530	NODE15	64555	220/DC	open,path	22-MAR-1991 18:57:33.
8074AB60	NODE16	64841	219/DB	open,path	25-MAR-1991 20:42:38.20

- **SHOW PORTS /ADDRESS=PE_PDT /NODE=nodename:** Displays the VC data for the specified remote node. This display is identical to that of the SHOW PORTS/VC=VC_nodename command.

VMS Version 5.4 Features

This appendix describes features introduced with VMS Version 5.4 but not yet documented in other printed manuals.

B.1 Summary of New VMS Version 5.4 Software Features

This section provides a summary (in Table B-1) of the VMS Version 5.4 software features. For information about new and enhanced hardware, see the *VMS Version 5.4 Release Notes*.

Table B-1 Summary of VMS Version 5.4 Software Features

VMS Version 5.4 Systemwide Features	
Vector Processing	Systemwide support for vector processing on VAX 9000 series and VAX 6000-400 series computers includes the VAX Vector Instruction Emulation Facility (VVIEF), specific DCL commands and lexical functions, and the Accounting, Error Log, Monitor, SDA, Debugger, Patch, and RTL MTH\$ facilities. See Section B.2 for a complete description of vector processing support.
DECdtm Services	Systemwide support for DECdtm services includes the Log Manager Control Program Utility (LMCP), MONITOR TRANSACTION command, new TRANSACTION_ID data type, and enhanced VMS RMS Journaling support. See Section B.3 for a complete description of DECdtm services.
VMS Version 5.4 General User Features	
DCL Commands	New and enhanced DCL commands let you control data compaction on tape drives that support data compaction, convert procedures written in PostScript to callable routines, compile fonts for the DECwindows server, and control and monitor specific processors and VAXft 3000 systems.
System Messages	Information about installing and accessing online help.
DECwindows User	You can now set another session language or change the target screen on the Session Manager; view PostScript files with the CDA Viewer; change to hexadecimal or octal mode in Calculator; use new File, Customize, and Help menus for interacting with Clock; and use DECwindows Mail to display PostScript files.

(continued on next page)

VMS Version 5.4 Features

B.1 Summary of New VMS Version 5.4 Software Features

Table B-1 (Cont.) Summary of VMS Version 5.4 Software Features

VMS Version 5.4 System Management Features	
AUTOGEN	This command procedure now includes support for parameter name validation, SYS\$SYSTEM:AGEN\$PARAMS.REPORT (a new file that replaces AGEN\$FEEDBACK.REPORT), reading external parameter files, controlling the size of page and swap files, new feedback parameters, new defined process logical names, a new technique for running AUTOGEN in batch mode, and the ability to use MAIL to send AGEN\$PARAMS.REPORT.
UETP	Enhancements to the User Environment Test Package include loading and testing of all installed and enabled vector processors, testing of the VAX Vector Instruction Emulation Facility (VVIEF), and support for the RRD40 compact disc drive, including Small Computer System Interface (SCSI) disk configurations.
SYSMAN Utility	Enhancements let you run a SYSMAN command procedure, define keys, spawn a subprocess, use DCL verification, and use loadable image commands.
VAXcluster Software	Enhancements include CI architecture extensions that allow multiple CI interfaces per CPU and multiple star couplers per VAXcluster system; MSCP server load sharing; and preferred path support for DSA disks (including RA-series disks and disks accessed through the MSCP server).
SYSGEN Utility	Enhancements include a new parameter for MicroVAX and VAXstation configurations that include third-party Small Computer System Interface (SCSI) devices, new parameters that support site-specific password policies, and new SHOW commands that display information such as bus identification statistics, device addresses mapped in the I/O space for the VAXBI bus, and device addresses mapped in the I/O space for the XMI bus.
Error Log Utility	Enhancements include support for VAXft 3000 device types, new device-class and entry-type keywords (to support vector processing and VAX 9000 systems) used with the /EXCLUDE and /INCLUDE qualifiers, and support for the new /NODE qualifier, which lets you produce a report of error log entries for specific nodes in a VAXcluster.
System Security	System security enhancements enable you to implement a site-defined password policy by screening new passwords and specifying password algorithms. This support includes enhancements to DCL commands, the SYSGEN Utility, the SYSMAN Utility, and system services. See Section B.10 for more information.
LMCP Utility	The new Log Manager Control Program Utility (LMCP) lets the system manager create and manage transaction log files in a DECdtm services environment. See Section B.11 for a complete description of this new utility.
Monitor Utility	Enhancements include support for vector processing with the new MONITOR VECTOR command and VECTOR class and support for DECdtm services with the new MONITOR TRANSACTION command and TRANSACTION class.

(continued on next page)

VMS Version 5.4 Features

B.1 Summary of New VMS Version 5.4 Software Features

Table B-1 (Cont.) Summary of VMS Version 5.4 Software Features

VMS Version 5.4 System Management Features	
NCP Utility	The Network Control Program Utility now includes support for a new line and circuit name specific to the VAXft 3000 system.
VMS Volume Shadowing	VMS Volume Shadowing phase II includes support for distributed, clusterwide shadowing of all MSCP-compliant DSA disks (with the same number of logical blocks) and shadowing of all DSA devices.
VMS Version 5.4 Programming Features	
VMS Debugger	Enhancements to the debugger's command and DECwindows interfaces let you debug programs containing VAX vector instructions.
Linker Utility	A new command line qualifier, /BPAGE, lets you specify larger page sizes.
Mail Utility Routines	New callable mail routines let you create applications that can perform a variety of Mail Utility functions and communicate with users on remote nodes connected to the system with DECnet-VAX.
System Services	New and enhanced system services support DECdtm services, system security enhancements, vector processing, volume shadowing, volume initialization, and the procedure for creating site-specific loadable images.
Run-Time Library	<p>New parallel processing (PPL\$) routines let you inform the PPL\$ facility when a new caller is forming or joining a parallel application, implement work queues, delete a PPL\$ application or object, set and adjust a semaphore maximum, disable event notification, or read a spin-lock state.</p> <p>New and enhanced mathematics routines (MTH\$) let you manipulate and perform operations on vectors.</p>
RMS	Enhancements provide asynchronous support for process-permanent files, an increase in the local buffer maximum, access-mode protection for RMS services and for specific data structures and their associated I/O buffers, and the ability for all applications to selectively suppress updates to the Expiration Date and Time, using XAB\$_NORECORD XABITM.
I/O Drivers	Enhancements include support for the pseudoterminal driver (FTDRIVER) and shadow set virtual driver (SHDRIVER), modifications to the item-list read function of the I/O status block (IOSB) and to the item-list terminal driver read verify operations for the TRIM\$_MODIFIERS item code, and the addition of three new ACP-QIO functions.

(continued on next page)

VMS Version 5.4 Features

B.1 Summary of New VMS Version 5.4 Software Features

Table B-1 (Cont.) Summary of VMS Version 5.4 Software Features

VMS Version 5.4 Programming Features	
System Dump Analyzer	New qualifiers to the SHOW PROCESS command let you display statistics about an image (/IMAGE) or about the values of the registers from the process's vector context area (/VECTOR_REGISTERS).
Device Support	Enhanced support includes VAX 9000 and VAX 6000 series systems. Programmers can write and debug driver software for non-Digital-supplied devices attached to a VAX 9000 system.
VAXTPU	Enhancements include work file support, a qualifier you can use to specify either character-cell or DECwindows interface, and new built-in procedures, including GET_INFO, that support journal recovery, pop-up menus, column context values for a buffer, markers within a buffer, and scrolling.
RMS Journaling	Enhancements support DECdtm services as well as existing applications and affect the Recovery Unit Facility (RUF), network support of remote files, RMS record streams, the RMS Detached Recovery server, placement of recovery unit journals, and access of files in a mixed-version cluster.
VMSINSTAL	A new data-file parameter (P4) in the Software Product Kit Building Procedure (SPKITBLD.COM) lets you specify the name of a data file. New callbacks affect messages displayed—and booting procedures required—during product installations and how you obtain a system-generated or installer-specified password.
DECwindows Programming	Enhancements include new programming examples in the DECW\$EXAMPLES directory, new support for the XUI Toolkit color mixing widget (both the Hue Lightness Saturation and Red, Green, Blue color models), support for the Display PostScript system (which provides text and image display capability for bitmapped workstations), and CDA Viewer support for PostScript files, Adobe Font metrics, and DECmath fonts.

B.2 Introduction to Vector Processing

The VMS Version 5.4 operating system supports vector processing on VAX 9000 series and VAX 6000-400 series computers. This section describes how vector processing works, how to manage resources, and how to write programs within a vector processing environment. The following sources in this appendix and in other documents also describe aspects of VMS Version 5.4 vector processing support:

- *VMS Version 5.4 Upgrade and Installation Manual* describe modifications to UETP.
- Section B.4.1 and the *VMS DCL Dictionary* describe new and modified DCL commands, qualifiers, and lexical functions.
- Section B.2.3.5 and the *VMS Debugger Manual* describe how to debug vectorized programs.
- Sections B.2.3.2, B.2.3.3, and B.2.3.4 describe new and modified system services.

- The *VMS RTL Mathematics (MTH\$) Manual* describes new and modified RTL mathematics routines.

B.2.1 Overview of the Vector Processing Environment

A single data item having one value is known as a **scalar**. A group of related scalar values, or elements, all of the same data type is known as a **vector**.

Traditional scalar computers operate only on scalar values and must process vector elements sequentially. Vector computers, on the other hand, recognize vectors as native data structures and can operate on an entire vector with a single vector instruction.

A vector processor can routinely process a vector four to five times faster than a traditional computer using only scalar instructions. Vector processors gain this speed advantage over scalar processors by their use of special hardware techniques designed for the fast processing of streams of data. These techniques include data pipelining, chaining, and other forms of hardware parallelism in memory and in arithmetic and logical functional units. Pipelined functional units allow the vector processor to overlap the execution of successive computations with previous computations. Chaining allows the results of one instruction to be forwarded to another before the first instruction has been completely processed.

B.2.1.1 VAX Vector Processing Systems

An extension to the VAX architecture defines an optional design for integrated vector processing that has been adopted by several VAX processing systems. The VAX vector architecture includes 16 64-bit vector registers (V0 through V15), each containing 64 elements; vector control registers, including the vector count register (VCR), vector length register (VLR), and vector mask register (VMR); vector functional units; and a set of vector instructions. VAX vector instructions transfer data between the vector registers and memory, perform integer and floating-point arithmetic, and execute processor control functions. A more detailed description of the VAX vector architecture, vector registers, and vector instructions appears in the *VAX MACRO and Instruction Set Reference Manual*.

Those VAX systems that comply with the VAX vector architecture are known as **vector-capable** systems.

A VAX vector processing system configuration includes one or more integrated scalar-vector processor pairs, or **vector-present processors**. Such a configuration can either be symmetric, including a vector coprocessor for each scalar, or asymmetric, incorporating additional scalar-only processors. Depending on the model of the VAX vector processing system, the scalar and vector CPUs of vector-present processors can be either a single, integral physical module or separate, physically independent modules. In either case the scalar and vector CPUs are logically integrated, sharing the same memory and transferring data over a dedicated, high-speed internal path. Because the CPUs are thus tightly coupled, use of the vector CPU foregoes the expense of I/O operations.

The scalar and vector CPUs operate asynchronously with respect to each other. The scalar CPU fetches and decodes all instructions issued by the current image and executes all scalar instructions. When it encounters a vector instruction, the scalar CPU passes it to the vector CPU. While the vector CPU is executing this instruction, the scalar CPU continues to fetch and decode instructions, executing any scalar instruction it encounters and sending any vector instructions it encounters to the vector CPU. The vector processor maintains a queue of pending instructions in which it places instructions it receives while it is busy. The VMS

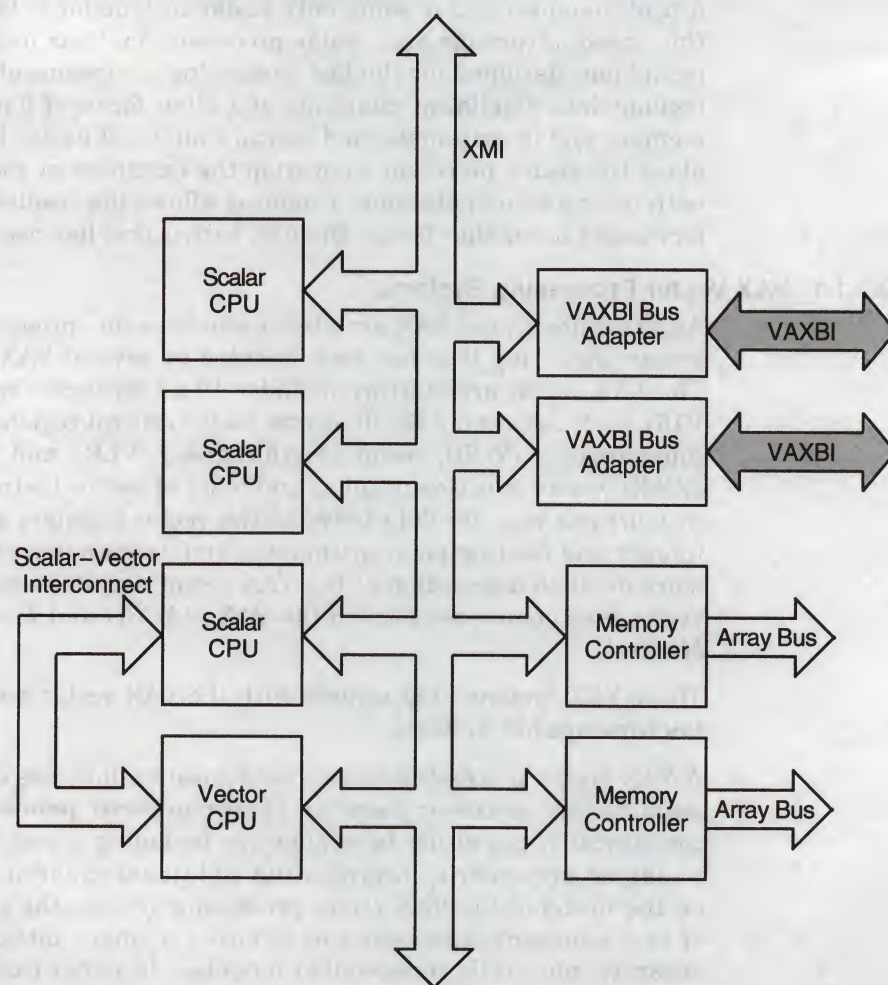
VMS Version 5.4 Features

VAX Vector Processing Systems

operating system and its vectorizing compilers help ensure that the activities of both scalar and vector CPUs are synchronized. (Section B.2.3.7 describes those situations in which vectorized VAX MACRO programs must enforce scalar and vector CPU synchronization.)

Certain VAX system models offer a vector processing option. In VAX 6000-400 series systems, the vector CPU occupies a slot on the memory interconnect; the scalar-vector interconnect joins it to the scalar CPU, which resides in an adjacent slot (see Figure B-1). In VAX 9000 series systems, the vector processor is an integral part of the CPU, as shown in Figure B-2.

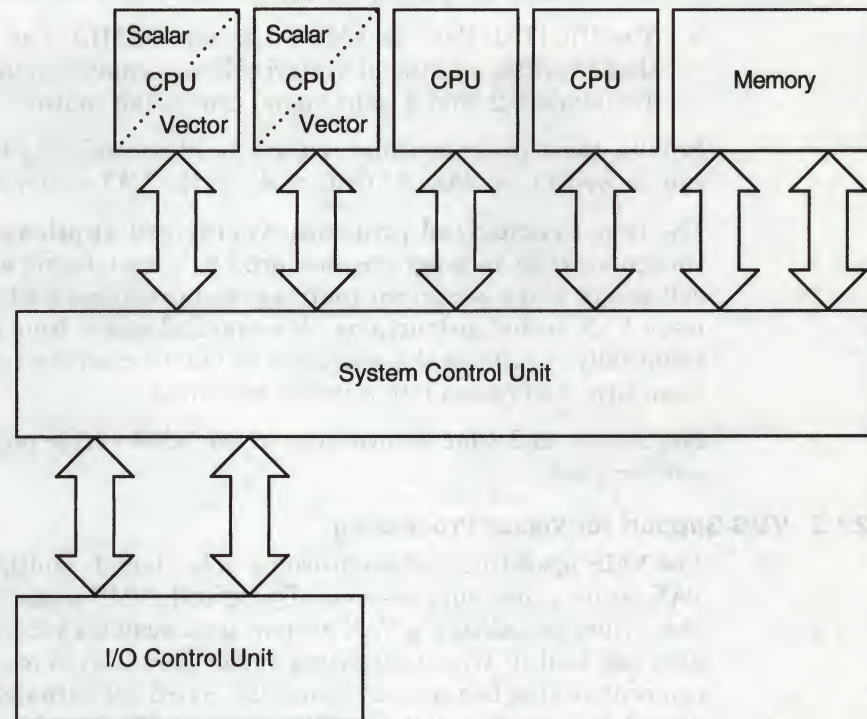
Figure B-1 VAX 6000-400 Series Vector-Present Processor Configuration



ZK-1945A-GE

Like VAX scalar processing systems, a VAX vector processing system can participate as a member of a VAXcluster or as a node in a network or it can be run as a standalone system.

Figure B-2 VAX 9000 Series Vector-Present Processor Configuration



ZK-1944A-GE

B.2.1.2 Vectorized Programs

The benefits of vectorization depend, to a large degree, on the specific techniques and algorithms of an application. CPU-intensive applications involving repeated operations on groups of simple elements are well-suited to vectorization. VAX vector processing systems are particularly beneficial in the fields of seismic analysis, weather forecasting, molecular modeling, computational fluid dynamics, signal processing, financial modeling, and finite element analysis.

There are several methods you can use to produce a vectorized program in a VMS system.

Most applications that benefit from vector processing can be developed as scalar programs in a high-level language and then submitted to a vectorizing compiler for that language. A **vectorizing compiler**, such as the VAX FORTRAN High Performance Option (HPO), can recognize sections of code within a program, usually inside formal loops, that can be vectorized. It analyzes data dependences, identifies other inhibitors to vector processing, and restructures code sequences to allow the compiler to generate optimized VAX vector instruction sequences.

Additionally, applications can be vectorized by a call to the vectorized routines in the VMS Run-Time Library mathematics facility (RTL MTH\$) or to the vectorized routines within the optional DIGITAL Extended Math Library (DXML):

- The vectorized RTL MTH\$ routines that can be called by a high-level language application include the Level 1 Basic Linear Algebra Subroutines (BLAS) and First-Order Linear Recurrence (FOLR) routines. In addition, VAX vectorizing compilers (and programs written in VAX MACRO) can generate calls to vectorized versions of the standard scalar RTL MTH\$ routines. (The

VMS Version 5.4 Features

Vectorized Programs

vectorized RTL MTH\$ routines are introduced in Section B.2.3.1 and fully discussed in the *VMS RTL Mathematics (MTH\$) Manual*.)

- The DIGITAL Extended Math Library (DXML) is an optional software product that provides additional vectorized mathematics routines such as BLAS Level 1-extended, 2, and 3, plus signal processing routines.

Finally, those programs that require strict control over the VAX vector hardware can be written in VAX MACRO and use the VAX vector instructions directly.

The terms **vectorized program**, **vectorized application**, and **vectorized image** all refer to programs produced by a vectorizing compiler, programs that call one or more vectorized routines, and programs written in VAX MACRO that issue VAX vector instructions. A vectorized image from any of these categories eventually results in the execution of one or more vector instructions that transform its process into a vector consumer.

See Section B.2.3 for an overview of the VMS vector processing programming environment.

B.2.1.3 VMS Support for Vector Processing

The VMS operating system provides fully shared, multiprogramming support for VAX vector processing systems. By default, VMS loads vector processing support code when initializing a VAX system that includes vector-present processors but does not load it when initializing vector-absent systems. (A system manager can control this behavior by using the SYSGEN parameter VECTOR_PROC, as described in Section B.2.2.1.) The presence of vector support code in a system has little effect on processes running in a scalar-only system or on scalar processes running in a vector-present system. If many processes must simultaneously compete for vector processor resources in a system, the system manager can maintain good performance by adjusting system resources and process quotas as indicated in Section B.2.2.3.1.

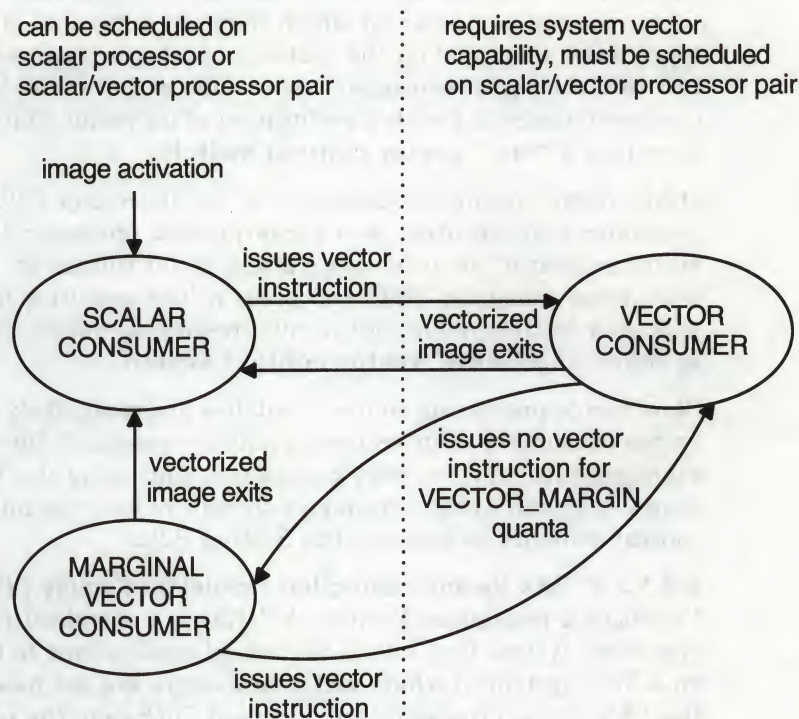
The VMS operating system makes the services of the vector processor available to system users by means of a software abstract known as a **capability**. A system manager can restrict the use of the vector processor to users holding a particular identifier by associating an access control list (ACL) with the vector capability object. See Section B.2.2.4 for additional information.

B.2.1.3.1 Life of a Vector Consumer As shown in Figure B-3, a process begins execution as a **scalar consumer**, partaking of the resources of a scalar processor or the scalar component of a vector-present processor.

When the image executing within the process's context issues its first vector instruction, VMS marks the process as requiring the system's vector capability. It also allocates sufficient system nonpaged dynamic memory in which to store this process's **vector context**. The vector context of a process consists of the contents of the vector registers V0 through V15, the contents of the vector control registers (VCR, VLR, and VMR), the vector processor status, and the vector exception state.

A process requiring the vector capability and having a vector context is known as a **vector consumer**. VMS *must* schedule a vector consumer on a vector-present processor. As long as it remains a vector consumer, a process is effectively prohibited from executing on any scalar processor in the system.

Figure B-3 Life of a Vector Consumer



ZK-1943A-GE

However, over the course of its execution, a typical vectorized image issues sequences of scalar instructions intermixed with sequences of vector instructions. For those periods in which it performs scalar operations exclusively, a process can relinquish its need for the vector capability and become eligible for execution on any processor in the system. VMS preserves the vector context of any such **marginal vector consumer** in the expectation that it will eventually issue another vector instruction and again become a vector consumer.

In a system in which many vector consumers are competing for the vector processor, the dynamic transition of vector consumers to marginal vector consumers (and back again) allows VMS to more efficiently distribute vector processor resources and enhances the performance of vectorized applications. Note that a system manager can control the transition of a vector consumer to a marginal vector consumer by setting the SYSGEN parameter VECTOR_MARGIN (as discussed in Section B.2.2.3.2).

Ultimately, a vector consumer or marginal vector consumer reverts to being a scalar consumer when the vectorized image it is executing exits.

In the course of system activity, another process could preempt the execution of a vector consumer on a vector-present processor. When this occurs, VMS immediately saves the vector consumer's scalar context, as it does for traditional scalar processes. However, VMS allows its vector context to remain intact in the vector CPU. Depending upon the nature of the intervening processes scheduled on that processor, VMS, in most cases, tries to reschedule a vector consumer on the vector-present processor on which it was last scheduled.

VMS Version 5.4 Features

VMS Support for Vector Processing

Because scalar consumers and marginal vector consumers do not use the vector CPU, they do not disturb the vector context of the latest vector consumer on the vector-present processor on which they are scheduled. If only processes of these types were scheduled on the vector-present processor since the vector consumer last ran, the vector consumer can resume execution on that processor without the overhead associated with a restoration of its vector context from memory. This is known as a **“fast” vector context switch**.

Other vector consumers, however, do use the vector CPU. When placing a vector consumer into execution on a vector-present processor, VMS stores in memory the vector context of the processor's latest vector consumer. When it later reschedules this vector consumer, VMS can place it into execution on any vector-present processor in the system, but it must restore its vector context from memory. This is known as a **“slow” vector context switch**.

Slow vector processing context switches are most likely when there are more vector consumers than vector-present processors in the systems. A system manager can adjust system parameters (including the VECTOR_MARGIN parameter) and system resources to help reduce the number of slow vector context switches as described in Section B.2.2.

B.2.1.3.2 VAX Vector Instruction Emulation Facility (VVIEF) The VAX Vector Instruction Emulation Facility (VVIEF) is a standard feature of the VMS operating system that allows vectorized applications to be written and debugged on a VAX system in which vector processors are not available. VVIEF emulates the VAX vector processing environment, including the nonprivileged VAX vector instructions and the VMS vector system services (described in Sections B.2.3.2, B.2.3.3, and B.2.3.4). Use of VVIEF is restricted to user-mode code.

VVIEF is strictly a program development tool and *not* a run-time replacement for vector hardware. There is no performance benefit from vectorizing applications to run under VVIEF; vectorized applications running under VVIEF typically execute five times slower than their scalar counterparts.

VMS supplies the VVIEF bootstrap code as an executive loadable image. The system manager invokes the command procedure SYS\$UPDATE:VVIEF\$INSTAL.COM to cause VMS to load VVIEF at the *next* system boot and each successive system boot. Note that, in the presence of VMS vector support code, VVIEF remains inactive. Although it is possible to prevent the loading of VMS vector support code in a vector-present system (see Section B.2.2.1) and activate VVIEF, there are few benefits. Should the only scalar-vector processor pair in the system fail, the execution of preempted vectorized applications will not be resumed under the emulator.

See Section B.2.2.6 for additional information on loading and unloading VVIEF.

B.2.2 Managing the Vector Processing Environment

Managing a VAX vector processing system includes the following tasks:

- Loading the VMS vector processing support code
- Configuring a vector processing system
- Managing processes requiring the system's vector processing resources
- Obtaining information about the status and use of the system's vector processing resources

- Loading the VAX Vector Instruction Emulation Facility (VVIEF) bootstrap code

This section describes the features VMS has introduced or enhanced to facilitate the accomplishment of these tasks. It concludes with a list of messages VMS uses to report information about the condition of the vector processing system.

B.2.2.1 Loading the VMS Vector Processing Support Code

By default, in a VAX vector processing system, VMS automatically loads the vector processing support code at boot time. You can override the default behavior by setting the static system parameter `VECTOR_PROC` as described in Table B-2.

Table B-2 Settings of VECTOR_PROC System Parameter

Value	Result
0	Do not load the vector processing support code, regardless of the system configuration.
1	Load the vector processing support code if there is at least one vector-present processor. This is the default value.
2	Load the vector processing support code if the system is vector capable. This setting is most useful for a system in which processors have separate power supplies. With this setting, you can reconfigure a vector processor into the system without rebooting the VMS operating system.

B.2.2.2 Configuring a VMS Vector Processing System

You can add or remove a vector-present processor to or from a VMS multiprocessing configuration at boot time by using the `SYSGEN` parameter `SMP_CPUS` or at run time by using the `DCL` commands `START/CPU` and `STOP/CPU`. Note that VMS treats the scalar and vector CPU components of a vector-present processor as a single processor, starting them and stopping them together.

At boot time, the setting of the `SYSGEN` parameter `SMP_CPUS` identifies which secondary processors in a VMS multiprocessing system are to be configured, including those processors that are vector present. (VMS always configures the primary processor.) The default value of `-1` boots all available processors, scalar and vector-present alike, into the configuration. (See the *VMS System Generation Utility Manual* for additional information about this parameter.) Note that, prior to starting a vector-present processor, you should make sure that the vector processing support code (see Section B.2.2.1) is loaded at boot time. Otherwise, processes will only be able to use the scalar CPU component of the vector-present processor.

To bring secondary processors into a running VMS multiprocessing system, you use the `DCL` command `START/CPU`. To remove secondary processors from the system, use the `STOP/CPU` commands. Again, you must make sure that the vector processing support code has been loaded at boot time for the vector CPU component of vector-present processors started in this way to be utilized.

However, note that if you enter a `STOP/CPU` command that would cause the removal of a vector-present processor that is the sole provider of the vector capability for currently active vector consumers, the command fails and generates a message. In extreme cases, such as the removal of a processor for repair, you can override this behavior by entering the command `STOP/CPU/OVERRIDE`. This command stops the processor, despite stranding processes.

When a STOP/CPU/OVERRIDE command is entered for a vector-present processor, or when a vector-present processor fails, VMS puts all stranded vector consumers into a CPU-capability-wait (RSN\$_CPUCAP) state until a vector-present processor is returned to the configuration. To any other process that subsequently issues a vector instruction (including a marginal vector consumer), VMS returns a "requested CPU not active" message (CPUNOTACT).

See the *VMS DCL Dictionary* for additional information about the START/CPU and STOP/CPU commands.

B.2.2.3 Managing Vector Processes

As described in Section B.2.1.3, VMS scheduling algorithms automatically distribute vector and scalar processing resources among vector consumers, marginal vector consumers, and scalar consumers. However, VAX vector processing configurations vary in two important ways:

- The amount of vector processing activity the configuration must accommodate
- The number of vector-present processors available in the configuration to service vector processing needs

In a configuration in which there are more vector consumers in a system than there are scalar-vector processor pairs to service them, vector consumers share vector-present processors according to process priority. At a given priority, VMS schedules vector consumers on a vector-present processor in a round-robin fashion. Each time VMS must schedule a new vector consumer on a vector-present processor, it must save the vector context of the current vector consumer in memory and restore the vector context of the new vector consumer from memory. When such "slow" vector context switches occur too frequently, a significant portion of the processing time is spent on vector context switches relative to actual computation.

Systems that have heavy vector processing needs should be adequately configured to accommodate those needs. There are, however, some mechanisms a system manager can use to tune the performance of an existing configuration.

B.2.2.3.1 Adjusting System Resources and Process Quotas Systems in which several vector consumers are active simultaneously might experience increased paging activity as processes share the available memory. To reduce process paging, you might need to use the Authorize Utility to adjust the working-set limits and quotas of the processes running vectorized applications. An increase of the process maximum working-set size (SYSGEN parameter WSMAX) might also be necessary. Additionally, a vectorized application can use the Lock Pages in Working Set system service (SYS\$LKWSET) to enhance its own performance.

VMS allots to each vector consumer 8 kilobytes of system nonpaged dynamic memory in which VMS stores vector context information. Depending on how many vector consumers are active in the system simultaneously, you might need to adjust the SYSGEN parameter NPAGEDYN. To determine the current usage of nonpaged pool, use the DCL command SHOW MEMORY/POOL/FULL, which displays the current size of nonpaged pool in bytes.

See the *VMS System Generation Utility Manual* and the *VMS Authorize Utility Manual* for additional information about these mechanisms.

To obtain optimal performance of a VAX vector processing system, you should take some care to set up generic batch queues that avoid saturating the system's vector resources. If a queue contains more active vectorized batch jobs than there are vector-present processors in the system, a significant portion of the processing time will be spent on vector context switches.

The recommended means for dispatching vectorized batch jobs to a VAX vector processing system is to set up a separate queue (for instance, VECTOR_BATCH) with a job limit equal to the number of vector-present processors in the system. When submitting vectorized batch jobs, users should be encouraged to submit them to this generic vector processing batch queue.

B.2.2.3.2 Distributing Scalar and Vector Resources Among Processes As a vector consumer, a process must be scheduled only on a vector-present processor. If the image the process is executing issues only scalar instructions for a period of time and must share the scalar-vector processor pair with other vector consumers, its inability to run on an available scalar processor could hamper its performance and the overall performance of the system.

By default, VMS assumes that, if a vector consumer has not issued a vector instruction for a certain period of time, it is unlikely that it will issue a vector instruction in the near future. VMS relinquishes this process's need for the vector capability, classifying it as a marginal vector consumer.

In an asymmetric vector processing configuration, detection of marginal vector consumers achieves the following desirable effects:

- Because a marginal vector consumer is eligible to run on a larger set of processors, its response time will improve.
- The scheduling of marginal vector consumers on scalar processors reduces the contention for vector-present processors.
- Because vector consumers issuing vector instructions are more likely to be scheduled on vector-present processors, the vector CPU is more efficiently used.

A system manager uses the SYSGEN parameter VECTOR_MARGIN to establish the interval of time at which VMS checks the status of all vector consumers. The VECTOR_MARGIN parameter accepts an integer value between 1 and -1 (FFFFFFFF₁₆). This value represents a number of consecutive process quanta (as determined by the SYSGEN parameter QUANTUM). If the process has not issued any vector instructions in the specified number of quanta, VMS declares it a marginal vector consumer. A value of -1 disables the checking mechanism.

The default value of the VECTOR_MARGIN parameter is 100₁₀.

B.2.2.4 Restricting Access to the Vector Processor by Using ACLs

Using the SET ACL and SHOW ACL commands, a system manager can restrict the use of the vector processor to users holding a particular identifier. By associating an access control list (ACL) with the vector capability, a university might limit use of the vector processor to faculty and students in an image processing course, or a service bureau might charge users for access to the vector capability, time spent on the vector processor, or both.

When using either the SET ACL or SHOW ACL command with Version 5.4 of the VMS operating system, the system manager can specify a new object type, CAPABILITY, as the argument to the /OBJECT_TYPE qualifier. This object type is a system capability, such as the ability to process VAX vector instructions. Currently, the only defined object name for the CAPABILITY object type is

VECTOR. Therefore, when using the SHOW ACL or SET ACL command, the system manager must supply the capability name (VECTOR) as the argument to the object type, as in the following examples. (For additional information about the SET ACL and SHOW ACL commands, see the *VMS DCL Dictionary*.)

Use the following DCL command format to establish one or more access control entries (ACEs) on the vector capability:

```
SET ACL/OBJECT=CAPABILITY VECTOR/ACL[=(ace[,...])]
```

Note that you must be in the SYSTEM user category (as described in *VMS DCL Concepts Manual*) to set an ACL on the vector capability.

The following DCL command displays the ACL on the vector capability:

```
$ SHOW ACL/OBJECT=CAPABILITY VECTOR
```

Note that the ACL is on the vector capability, not on the use of any or all vector-present processors in the system. For this reason, VMS can still schedule processes without permission to use the vector capability on a vector-present processor. However, these processes can use only the scalar CPU component of the processor and cannot execute vector instructions. Likewise, because the ACL is on the vector capability and not on a vector-present processor, you cannot establish an ACL to force long-running jobs to a specific processor.

The Change ACL (\$CHANGE_ACL) and Check Access (\$CHECK_ACCESS) system services provide means for setting and removing ACLs on the VECTOR capability and for checking a process's ability to use vector processing resources.

B.2.2.5 Obtaining Information About a Vector Processing System

You can obtain information about the status of the vector processing system and the use of the system by individual processes through various means, including:

- The DCL lexical functions F\$GETJPI and F\$GETSYI
- The DCL command SHOW CPU
- The DCL commands SHOW PROCESS and LOGOUT/FULL
- The Accounting Utility (ACCOUNTING)
- The Error Log Utility (ERROR LOG)
- The Monitor Utility (MONITOR)

B.2.2.5.1 DCL Lexical Functions F\$GETJPI and F\$GETSYI The DCL lexical function F\$GETJPI accepts the following items and returns the corresponding information regarding the vector status of a specified process:

Item	Return Type	Information Returned
FAST_VP_SWITCH	Integer	Number of times this process has issued a vector instruction that resulted in an inactive vector processor being enabled without the expense of a vector context switch
SLOW_VP_SWITCH	Integer	Number of times this process has issued a vector instruction that resulted in an inactive vector processor being enabled with a full vector context switch
VP_CONSUMER	Boolean	Flag indicating whether the process is a vector consumer

VMS Version 5.4 Features

Obtaining Information About a Vector Processing System

Item	Return Type	Information Returned
VP_CPUTIM	Integer	Total amount of time the process has accumulated as a vector consumer

The DCL lexical function F\$GETSYI accepts the following items and returns the corresponding information regarding the status of the vector processing system:

Item	Return Type	Information Returned
VP_NUMBER	Integer	Number of vector processors in the system
VP_MASK	Integer	Mask indicating which processors in the system have vector coprocessors
VECTOR_EMULATOR	Integer	Flag indicating the presence of the VAX Vector Instruction Emulation Facility (VVIEF) in the system

See the *VMS DCL Dictionary* for additional information about the DCL lexicals F\$GETJPI and F\$GETSYI.

B.2.2.5.2 SHOW CPU Command The SHOW CPU/FULL command lists the capabilities of the specified CPU. The manager of a VAX vector processing system can issue this command to determine the presence of the vector capability in the system prior to executing a STOP/CPU command.

See the *VMS DCL Dictionary* for additional information about the SHOW CPU command.

B.2.2.5.3 SHOW PROCESS and LOGOUT/FULL Commands If the target process has accrued any time as a vector consumer scheduled on a vector-present processor, the DCL commands SHOW PROCESS and LOGOUT/FULL display the elapsed vector CPU time and the charged vector CPU time, respectively.

To accumulate vector CPU time, a process must be a vector consumer (that is, require the system vector capability) and be scheduled on a vector-present processor. VMS still charges the vector consumer vector CPU time, even if, when scheduled on the vector-present processor, it does not actually use the vector CPU. Note that, because scalar consumers and marginal vector consumers do not use the vector CPU, they do not accrue vector CPU time, even when scheduled on a vector-present processor.

See the *VMS DCL Dictionary* for additional information about the SHOW PROCESS and LOGOUT commands.

B.2.2.5.4 Vector Processing Support Within the VMS Accounting Utility (ACCOUNTING) In its full listing format, the VMS Accounting Utility displays the vector CPU time accumulated by a process or an image during its life span.

A process accumulates vector CPU time while it is a vector consumer (that is, requiring the system vector capability) and it is scheduled on a vector-present processor. VMS still charges the vector consumer vector CPU time, even if, when scheduled on the vector-present processor, it does not actually use the vector CPU. Note that, because scalar consumers and marginal vector consumers do not use the vector CPU, they do not accrue vector CPU time, even when scheduled on a vector-present processor.

VMS Version 5.4 Features

Obtaining Information About a Vector Processing System

An image accrues vector CPU time while it is executing within the context of a vector consumer on a vector-present processor. Because VMS marks all processes, including vector consumers, as scalar consumers at image rundown, it is impossible for an image that issues only scalar instructions to accumulate vector CPU time.

The /SORT qualifier to the ACCOUNTING command accepts the VECTOR_PROCESSOR keyword and sorts the accounting records according to ascending or descending vector CPU time. The /REPORT qualifier also accepts the VECTOR_PROCESSOR keyword and produces a summary report of vector CPU usage.

See Section B.2.3.8 for a description of the vector CPU time field in the ACCOUNTING resource packet. The *VMS Accounting Utility Manual* provides a complete description of the VMS Accounting Utility.

B.2.2.5.5 Vector Support Within the Error Log Utility (ERROR LOG) With Version 5.4 of the Error Log Utility, the /INCLUDE qualifier to the ANALYZE /ERROR_LOG command accepts the device-class keyword VECTOR, which produces an error log report that includes vector processing errors. (Specifying the VECTOR keyword with the /EXCLUDE qualifier excludes vector processing errors from the error log report.)

B.2.2.5.6 Vector Support Within the VMS Monitor Utility (MONITOR) With Version 5.4 of the VMS Monitor Utility, the new MONITOR VECTOR command initiates monitoring of the VECTOR class and displays the number of 10-millisecond clock ticks per second in which one or more vector consumers have been scheduled on each currently configured vector processor.

See Section B.12.3 for a complete description of the MONITOR VECTOR command and the VECTOR class. See Section B.2.3.9 and Section B.12.4 for related information about the VECTOR class record and format. Refer to the *VMS Monitor Utility Manual* if you need additional information about the VMS Monitor Utility.

B.2.2.6 Loading the VAX Vector Instruction Emulation Facility (VVIEF)

The VAX Vector Instruction Emulation Facility (VVIEF) is a standard feature of the VMS operating system that allows vectorized applications to be written and debugged on a VAX system in which vector processors are not available. VVIEF is intended strictly as a program-development tool and *not* as a run-time replacement for vector hardware. There is no performance benefit from vectorizing applications to run under VVIEF; vectorized applications running under VVIEF typically execute five times slower than their scalar counterparts.

VMS supplies the VVIEF bootstrap code as an executive loadable image. To cause VMS to load VVIEF at the *next* system boot and at each subsequent system boot, invoke the command procedure SYS\$UPDATE:VVIEF\$INSTAL.COM. To unload VVIEF, invoke the command procedure SYS\$UPDATE:VVIEF\$DEINSTAL.COM and reboot the system. You can determine the presence or absence of VVIEF on a system by issuing the following DCL commands:

```
$ X = F$GETSYI("VECTOR_EMULATOR")
$ SHOW SYMBOL X
X = 1   Hex = 00000001   Octal = 0000000001
```

A return value of 1 indicates the presence of VVIEF; a value of 0 indicates its absence.

Note that, although VVIEF might be loaded into the system, in the presence of VMS vector support code, it remains inactive. Although it is possible to prevent the loading of VMS vector processing support code in a vector-present system (see Section B.2.2.1) and activate VVIEF, there are few benefits. Should the only vector-present processor in the system fail, the execution of preempted vectorized applications will not resume under the emulator.

B.2.2.7 System Messages Related to Vector Processing Activities

Table B-3 lists the system messages that might result from vector activity in a VAX vector processing system. It describes the conditions that might have resulted in the message and suggests how you can repair the condition causing the error.

For information about how VMS reports exception conditions to condition handlers, see Section B.2.3.6.

Table B-3 System Messages Relating to Vector Processing

Message	Message Text	Description and Recovery
ACCVIO	access violation, reason mask = xx, virtual address = location, PC = location, PSL = xxxxxxx	See the <i>VMS System Messages and Recovery Procedures Reference Manual</i> for a description of the ACCVIO message. The lowest three bits of the reason mask indicate that an instruction has caused a length violation (bit 0), referenced the process page table (bit 1), and attempted a read or modify operation (bit 2). VMS defines two additional bits to reflect vector processing memory management exceptions: a vector operation on an improperly aligned vector element in memory (bit 3) and vector instruction reference to an I/O space address (bit 4).
BADCONTEXT	invalid or corrupted context encountered	The vector state of a mainline routine as saved in process P1 space has been corrupted and cannot be restored. A call to the Restore Vector State system service (SYS\$RESTORE_VP_STATE) can result in this error, if some coding error has overwritten the saved vector state. (See the <i>VMS System Services Reference Manual</i> for more information about this system service.)
CPUNOTACT	requested CPU not active	The current process requires system capabilities that are not available or no longer available among the active processors in the system. If this message is associated with a vector disabled (VECDIS) status code, a vector-present processor within the system is not available, has failed, or has been removed from the system. See Section B.2.2.2.
EXQUOTA	exceeded quota	If this message is associated with a vector disabled (VECDIS) status code, the process's paging file quota prohibits the allocation of sufficient process memory for storing its mainline vector state. (See Section B.2.2.3.1.)

(continued on next page)

VMS Version 5.4 Features

System Messages Related to Vector Processing Activities

Table B-3 (Cont.) System Messages Relating to Vector Processing

Message	Message Text	Description and Recovery
ILLVECOP	illegal vector opcode fault, opcode='xx', PC='location', PSL='xxxxxxx'	An operation code designated as an illegal vector opcode by the VAX architecture has been encountered during the execution of an image. See Section B.2.3.6 and the <i>VAX MACRO and Instruction Set Reference Manual</i> for additional information about this exception.
IMGVEXC	image exiting with pending vector exceptions	An exception has resulted due to the execution of a vector instruction issued by an image, but the image has exited before the exception could be delivered. See Section B.2.3.7.4.
INSFMEM	insufficient dynamic memory	If this message is associated with a vector disabled (VECDIS) status code, the current process has issued a vector instruction, but insufficient system nonpaged dynamic memory exists to establish the process as a vector consumer. (See Section B.2.2.3.1.)
INSFWSL	insufficient working-set limit	If this message is associated with a vector disabled (VECDIS) status code, the process's current working-set list limit does not allow its mainline vector state to be resident in memory. (See Section B.2.2.3.1.)
NOPRIV	no privilege for attempted operation	If this message is associated with a vector disabled (VECDIS) status code, an ACL on the system's vector capability has prevented the process from executing vector instructions. (See Section B.2.2.4.)
NOSAVPEXC	no saved vector exception for the exception-id	A call was made to the Restore Vector Processing State system service (SYS\$RESTORE_VP_EXCEPTION) that specified a value for an exception ID that does not correspond to that of any saved vector exception state. (See the <i>VMS System Services Reference Manual</i> for more information about this system service.)

(continued on next page)

Table B-3 (Cont.) System Messages Relating to Vector Processing

Message	Message Text	Description and Recovery
VARITH	vector arithmetic fault, summary=xx, mask=xx, PC=location, PSL=xxxxxxx	<p>A vector operate instruction, executing within the current context, has resulted in a vector arithmetic trap. (See Section B.2.3.6 for assistance in interpreting the exception summary mask, vector register mask, PC, and PSL.)</p> <p>Because arithmetic operations are performed in a substantially different manner on vectors than on scalars, the resolution of vector arithmetic exceptions requires some special techniques. (See Section B.2.3.6 for information about the mechanisms by which exceptions are reported and identified.) One or a combination of several debugging strategies can help you determine which calculations resulted in the reported error or errors:</p> <ul style="list-style-type: none"> • Recompile the source with the /DEBUG, /NOVECTOR, /CHECK=BOUNDS qualifiers; relink using the /DEBUG and /MAP qualifiers; and run the resulting scalar image with the /DEBUG qualifier. A scalar arithmetic exception should occur at the calculation that caused the original vector arithmetic exception. • Recompile the source using the /DEBUG, /LIST, and /VECTOR qualifiers; relink using the /DEBUG and /MAP qualifiers; and run the resulting image with the /DEBUG qualifier. (If the /ASSUME=NOACCURACY_SENSITIVE qualifier was used in the original compilation, specify /ASSUME=ACCURACY_SENSITIVE.) Use the SET VECTOR_MODE SYNCHRONIZED or the SYNCHRONIZE VECTOR_MODE debugger command to guarantee that all exceptions resulting from vector operations be delivered before the execution of the next scalar instruction. Step through the program, inspecting the contents of those vector registers that are involved in each vector operation.

(continued on next page)

VMS Version 5.4 Features

System Messages Related to Vector Processing Activities

Table B-3 (Cont.) System Messages Relating to Vector Processing

Message	Message Text	Description and Recovery
		<p>When a vector operate instruction causes a floating-point exception in a vector element, the exception result is encoded into the corresponding element of the destination register. When the destination vector register is the target of an EXAMINE/FLOAT debugger command, the debugger displays the decoded exception message in the associated vector element.</p> <p>When a vector operate instruction causes an integer overflow in a vector element, the corresponding element of the destination register contains a value that is larger than 32 bits, but of a different sign than the instruction's operands. When the destination vector register is the target of an EXAMINE debugger command, you must inspect each element for such results.</p>
VASFUL	virtual address space is full	<p>If this message is associated with a vector disabled (VECDIS) status code, insufficient process virtual address space exists to allow the current process's mainline vector state to be saved. (See Section B.2.2.3.1.)</p>
VECALIGN	access violation, reason mask = xx, virtual address = location, PC = location, PSL = xxxxxxxx	<p>The current process has issued a VAX vector memory access instruction that has attempted an operation on an improperly aligned vector element. The VAX architecture requires that vector operands to vector memory access instructions be naturally aligned in memory. Longwords must be aligned on longword boundaries; quadwords must be aligned on quadword boundaries. See Section B.2.3.6 and the <i>VAX MACRO and Instruction Set Reference Manual</i> for additional information.</p>

(continued on next page)

Table B-3 (Cont.) System Messages Relating to Vector Processing

Message	Message Text	Description and Recovery
VECDIS	vector disabled fault, code=xx, PC = location, PSL = xxxxxxxx	<p>The current process has issued a vector instruction that requires that a vector processor become active. Under normal circumstances, this event is not reported to a system user. However, if the vector processor was unavailable due to some previously unreported condition, the VECDIS message is issued in association with one of the following messages.</p> <ul style="list-style-type: none"> • BADCONTEXT • CPUNOTACT • EXQUOTA • INSFMEM • INSFWSL • MCHECK • NOPRIV • VASFUL <p>See the description of the associated message in this table and the <i>VMS System Messages and Recovery Procedures Reference Manual</i> for additional information about any specific error.</p>

B.2.3 Programming in a Vector Processing Environment

Most applications that benefit from vector processing can be developed as scalar programs in a high-level language and then submitted to a vectorizing compiler for that language.

Additionally, applications can be vectorized by a call to the vectorized routines in the VMS Run-Time Library mathematics facility (RTL MTH\$) or to the vectorized routines within the optional DIGITAL Extended Math Library (DXML):

- The vectorized RTL MTH\$ routines that can be called by a high-level language application include the Level 1 Basic Linear Algebra Subroutines (BLAS) and First-Order Linear Recurrence (FOLR) routines. In addition, VAX vectorizing compilers (and programs written in VAX MACRO) can generate calls to vectorized versions of the standard scalar RTL MTH\$ routines. (The vectorized RTL MTH\$ routines are introduced in Section B.2.3.1 and fully discussed in the *VMS RTL Mathematics (MTH\$) Manual*.)
- The DIGITAL Extended Math Library (DXML) is an optional software product that provides additional vectorized mathematics routines such as BLAS Level 1-extended, 2, and 3, plus signal processing routines.

Finally, those programs that require strict control over the VAX vector hardware can be written in VAX MACRO and use the VAX vector instructions directly.

Use of high-level interfaces to VAX vector processing systems, such as the VAX FORTRAN HPO vectorizing compiler and the vectorized RTL MTH\$ routines, provide a mechanism for quickly developing a vectorized program that conforms to the requirements of the VAX Procedure Calling and Condition Handling Standard and the VAX vector architecture. For instance, VAX vectorizing

VMS Version 5.4 Features

Programming in a Vector Processing Environment

compilers and vectorized library routines automatically handle the complexities of properly handling scalar-vector synchronization, vector memory alignment, and the preservation of vector state across procedure calls. Additionally, the VAX FORTRAN HPO vectorizing compiler can recognize sections of code within a program, usually inside formal loops, that can be vectorized. It analyzes data dependences, identifies inhibitors to vector processing, and restructures code sequences to allow the compiler to generate optimized VAX vector instruction sequences.

By contrast, VAX MACRO programmers must themselves ensure that vector code conforms to the rules stated in the *VAX MACRO and Instruction Set Reference Manual* and Section B.2.3.7.

If you must write a vectorized program in VAX MACRO, you should be aware of the following:

- You must specifically enable the processing of vector instructions by the VAX MACRO assembler by assembling with the `/ENABLE` or `/NODISABLE` qualifier to the MACRO command and supplying the keyword `VECTOR`. You can also explicitly enable the assembly of vector instructions by using the `.ENABLE VECTOR` directive.
- The VAX MACRO assembler parses the assembler notation form of vector instructions and produces binary code in the instruction stream form prescribed by the VAX vector architecture. The *VAX MACRO and Instruction Set Reference Manual* describes both vector instruction forms and presents the assembler notation form in its instruction pages.
- VAX MACRO programs must synchronize the vector CPU's memory references across procedure calls, as well as guarantee that pending vector exceptions are raised before crossing procedure boundaries. VAX MACRO programs must also ensure that the vector CPU's memory references are synchronized with the scalar CPU's memory references. Section B.2.3.7 and the *VAX MACRO and Instruction Set Reference Manual* describe the mechanisms by which VAX MACRO code can comply with these requirements.
- The *VAX MACRO and Instruction Set Reference Manual* lists several additional restrictions, including the following:
 - VAX MACRO programs must naturally align vector operands to vector memory access instructions. Longwords must be aligned on longword boundaries; quadwords must be aligned on quadword boundaries.
 - VAX MACRO instructions cannot reference addresses in I/O space.
 - Vector instructions cannot be issued at elevated interrupt priority levels (IPLs), specifically at or above `IPL$_RESCHED`. The vector disabled handler will force a system crash with the `VPIPLHIGH` bugcheck code ("IPL too high to use the Vector Facility") when a user vector instruction is issued at or above `IPL$_RESCHED`.

The remainder of this section discusses the following topics:

- Vector routines in the MTH\$ Run-Time Library
- Obtaining information about a vector processing system
- Releasing the vector processor
- Preserving and restoring a routine's vector state
- Issuing vector instructions at high IPLs

- Debugging a vector application
- Servicing vector processing exceptions
- Utilizing vector information contained within the informational packets generated by the VMS Accounting Utility and VMS Monitor Utility

B.2.3.1 Vector Routines in the MTH\$ Run-Time Library

The RTL MTH\$ facility provides three sets of routines that allow manipulation of vectors and perform operations on vectors:

- The Basic Linear Algebra Subroutines (BLAS) Level 1 copy vectors swap the elements of two vectors, scale vector elements, perform reduction operations on vectors, and effect a Givens plane rotation. Scalar and vector versions of the BLAS Level 1 are provided in the new BLAS1\$ and VBLAS1\$ facilities, respectively. BLAS Level 1 forms an integral part of many standard libraries such as LINPACK and EISPACK. The version of the subroutines in the RTL VBLAS1\$ facility have been tuned to the VAX architecture to take advantage of vectorization.
- The First Order Linear Recurrence (FOLR) routines provide a vectorized algorithm for the linear recurrence relation. (The traditional algorithm generally inhibits vectorization by using the result of a previous pass through a loop as an operand in subsequent passes through the loop.)

The FOLR routines in the RTL MTH\$ facility perform addition, multiplication, or both addition and multiplication, on recursion elements, saving the result of each iteration in an array or saving only the last result in a variable. The RTL MTH\$ facility supplies these routines in four groups, each accepting any of four data types: longword integer, F-floating, D-floating, or G-floating.

- Certain key MTH\$ routines have been vectorized to support Digital's vectorizing compilers, such as the VAX FORTRAN HPO. Vectorized versions of key F-floating, D-floating, and G-floating scalar routines employ vector hardware to the fullest, while maintaining results that are identical to those of their scalar counterparts.

Vectorized MTH\$ routines are never called directly from a high-level language program. At a call to a scalar version of one of these routines, a vectorizing compiler automatically determines whether an operation should be performed by the vector or scalar version of a routine. VAX MACRO programs, however, can call the vectorized MTH\$ routines directly.

See the *VMS RTL Mathematics (MTH\$) Manual* for complete information about these routines.

Note that the VAX FORTRAN HPO detects usage of the vectorizable constructs within source code and automatically issues a call to the appropriate RTL MTH\$ routines. See the description of the /BLAS qualifier in the compiler documentation.

B.2.3.2 Obtaining Information About a Vector Processing System

The Get Job/Process Information system service (SYS\$GETJPI) accepts the following item codes and returns the indicated information about the vector status of one or more processes in the system:

Item Code	Return Value
JPI\$FAST_VP_SWITCH	Unsigned longword containing the number of times this process has issued a vector instruction that resulted in an inactive vector processor being enabled without the expense of a vector context switch. This count reflects those instances in which the process has reenabled a vector processor on which the process's vector context has remained intact.
JPI\$SLOW_VP_SWITCH	Unsigned longword containing the number of times this process has issued a vector instruction that resulted in an inactive vector processor being enabled with a full vector context switch. This vector context switch involves the saving of the vector context of the process that last used the vector processor and the restoration of the vector context of the current process.
JPI\$VP_CONSUMER	Byte, the low-order bit of which, when set, indicates that the process is a vector consumer.
JPI\$VP_CPUTIM	Unsigned longword that contains the total amount of time the process has accumulated as a vector consumer.

The Get Systemwide Information system service (SYS\$GETSYI) accepts the following item codes and returns the indicated information about the vector status of the system:

Item Code	Return Value
SYI\$VP_NUMBER	Unsigned longword containing the number of vector processors in the system.
SYI\$VP_MASK	Longword mask, the bits of which, when set, indicate which processors in the system have vector coprocessors.
SYI\$VECTOR_EMULATOR	Byte, the low-order bit of which, when set, indicates the presence of the VAX Vector Instruction Emulation Facility (VVIEF) in the system.

See the *VMS System Services Reference Manual* for additional information about the \$GETJPI and \$GETSYI system services.

B.2.3.3 Releasing the Vector Processor

The Release Vector Processor system service (SYS\$RELEASE_VP) terminates the current process's status as a vector consumer. Because \$RELEASE_VP declares that the process no longer needs the system's vector capability, VMS is no longer restricted to scheduling it on a vector-present processor. As a result, the process can be placed into execution on other CPUs in the system.

See the *VMS System Services Reference Manual* for a full description of the invocation format and functions of this service.

B.2.3.4 Preserving and Restoring a Routine's Vector State

The vector context of a process consists of the contents of the vector registers V0 through V15, the contents of the vector control registers (VLR, VCR, and VMR), the vector processor status, and the vector exception state. When a vectorized application involves calls among two or more routines, each of which issues vector instructions, two components of a process's vector context must be considered:

- The vector registers that are shared across procedure calls
- The vector exception state that exists just prior to a procedure call or return

The VAX Procedure Calling and Condition Handling Standard (see Section B.2.3.7.1) requires that calling and called procedures agree as to the conventions by which they preserve and manipulate vector registers. For languages such as VAX MACRO, which allows direct access of registers, either the calling procedure or called procedure can save or restore vector registers shared between routines.

The standard also requires that, if a procedure executes a vector instruction that might possibly raise an exception, the procedure must ensure that this exception is reported before it calls another procedure, returns to its caller, or exits. If a vector exception were pending at the time a procedure transferred control, it would be reported in the context of a procedure that did not incur the exception. VAX vectorizing compilers ensure that compiled code properly follows this requirement; calls to vector routines in the RTL MTH\$ facility (as described in Section B.2.3.1) also comply with this prescription. However, vectorized code written in VAX MACRO must adhere to the rules discussed in Section B.2.3.7.4.

For those routines that can run asynchronously with respect to the mainline routine—such as asynchronous system trap (AST) routines, condition handlers, and exit handlers—VMS automatically handles the saving and restoring of vector context. VMS supports vector usage in these asynchronous routines by providing each routine that is active asynchronously within a process with its own **vector state**.

The vector state of a routine reflects the vector context of the process at the time of the routine's execution or preemption, as the case may be, when an AST is delivered to the process or a condition handler is triggered. A process can have several vector states; for instance, one for its mainline routine and one for an AST routine that has interrupted the mainline. However, a process has only a single vector context, reflecting its current vector state.

VMS automatically preserves the vector state of a routine as follows:

- When a user-mode AST routine issues a vector instruction, VMS saves the vector state of the mainline routine. It restores the mainline vector state when the AST routine exits.
- When a user-mode condition handler issues a vector instruction, VMS saves the vector state of the mainline routine. It restores the mainline vector state on continuing from the exception and on stack unwind.
- When calling an exit handler, VMS clears the vector exception state.

By default, when an asynchronous routine interrupts the execution of a mainline routine, VMS creates a new vector state when the routine issues its first vector instruction. At this point, the vector state of the mainline routine is inaccessible to the asynchronous routine.

In certain cases, however, an AST routine or condition handler might need to read or modify the saved exception state of the mainline routine. To do so, the routine must call the Restore Vector State system service (SYS\$RESTORE_VP_STATE). \$RESTORE_VP_STATE restores the vector state of the process's mainline routine.

In very rare cases, a procedure might need to preserve and restore the current vector exception state across individual contexts that it creates and maintains. For instance, a task manager could set up several discrete tasks, each of which has its own vector state. To implement such a system, the routine saves the contents of the appropriate vector registers and calls the Save Vector Exception State (SYS\$SAVE_VP_EXCEPTION) and Restore Vector Exception State (SYS\$RESTORE_VP_EXCEPTION) system services.

The Save Vector Exception State service saves in memory any pending vector exception state and clears the vector processor's current exception state. The Restore Vector Exception State service restores from memory the vector state saved by a prior call to \$SAVE_VP_EXCEPTION. After a routine invokes this service, the next vector instruction issued within the process causes the restored vector exception to be reported.

See the *VMS System Services Reference Manual* for a full description of the syntax and use of the \$SAVE_VP_EXCEPTION, \$RESTORE_VP_EXCEPTION, and \$RESTORE_VP_STATE system services.

B.2.3.5 Debugging a Vectorized Program

Version 5.4 of the VMS operating system supports the debugging of vector applications by adding new capabilities to the VMS Debugger, the VMS System Dump Analyzer (SDA), the debuggers of the VMS Delta/XDelta Utility (DELTA/XDELTA), and the Patch Utility. Additionally, the VMS exception detecting and reporting mechanism collects information regarding the nature and context of vector processing errors. Section B.2.3.6 describes the information VMS provides when reporting a vector processing exception.

B.2.3.5.1 Vector Processing Support Within the VMS Debugger Through enhancements and additions to its existing command set, the VMS Debugger allows you to correct and tune vectorized applications. VMS Debugger commands enable you to perform the following tasks:

- Control and monitor the execution of vector instructions with breakpoints, watchpoints, and other mechanisms
- Examine and deposit into the vector control registers (VCR, VLR, and VMR) and the vector registers (V0 through V15)
- Examine and deposit vector instructions
- Perform masked operations on vector registers to display only certain register elements or override the masking associated with a vector instruction
- Control synchronization between the scalar and vector processors
- Save and restore the current vector state when using the CALL command to execute a routine that might affect the vector state
- Display vector register data using a screen-mode display
- Display the decoded results of vector arithmetic exceptions

See the *VMS Debugger Manual* for complete information about these and other functions of the VMS Debugger.

B.2.3.5.2 Vector Processing Support Within the VMS System Dump Analyzer

(SDA) The System Dump Analyzer (SDA) provides several mechanisms for examining vector instructions and vector context from a system dump file or in a running system. They include the following:

- You can decode and display vector instructions using the **EXAMINE /INSTRUCTION** command. This command displays the vector opcodes, switches, and operands in the form and order defined by the **VAX MACRO** assembler notation. Note that, when you use SDA to display the contents of memory locations, vector instructions appear in the instruction stream format defined by the VAX architecture; that is, an opcode followed by the vector control word in immediate addressing mode. (See the *VAX MACRO and Instruction Set Reference Manual* for descriptions of the assembler notation and instruction stream formats of vector instructions.)
- You can examine the values of a process's vector registers and vector control registers by entering the **SHOW PROCESS/VECTOR_REGISTERS** command. This command obtains the values of the registers from the process's vector context area. Note that the names of these registers (**V0** through **V15**, **VCR**, **VLR**, and **VMR**) are not defined in the SDA symbol table. You cannot display the current contents of any of these registers using the **EXAMINE** or **EVALUATE** command.
- You can format the contents of a memory location as a process's vector context block. The symbol table **SYS\$SYSTEM:SYSDEF.STB** contains a definition of this structure. You must use the **READ** command to load the symbols defined within this file into the SDA symbol table.
- You can determine the presence and location of the VMS vector processing support code (**VECTOR_PROCESSING.EXE**) and the VAX Vector Instruction Emulation Facility (**VVIEF**) bootstrap code (**VVIEF\$BOOTSTRAP.EXE**) by entering the SDA command **SHOW EXECUTIVE**. Both are executive loadable images. You can also use the SDA command **READ/EXECUTIVE** to load definitions of locations within these images into the SDA symbol table.

B.2.3.5.3 Vector Processing Support Within the VMS Delta/XDelta Utility The VMS Delta/XDelta Utility (**DELTA/XDELTA**) provides mechanisms for stepping through vector code, examining and decoding vector instructions, and setting breakpoints at vector instructions. You can use the following commands to debug a vectorized application:

- The **Open Location and Display Instruction in Instruction Mode** command (!) displays the vector opcodes, switches, and operands in the form and order defined by the **VAX MACRO** assembler notation. Note that, when you use **DELTA/XDELTA** to display the contents of memory locations, vector instructions appear in the instruction stream format defined by the VAX architecture; that is, an opcode followed by the vector control word in immediate addressing mode. (See the *VAX MACRO and Instruction Set Reference Manual* for descriptions of the assembler notation and instruction stream formats of vector instructions.)
- The **Step Instruction** command (S) enables you to single-step through vector instructions.
- The **List Names and Locations of Loaded Executive Images** command (;L) enables you to determine the presence and location of the VMS vector processing support code (**VECTOR_PROCESSING.EXE**) and

VMS Version 5.4 Features

Debugging a Vectorized Program

the VAX Vector Instruction Emulation Facility (VVIEF) bootstrap code (VVIEF\$BOOTSTRAP.EXE).

- The Breakpoint (;B) and Proceed from Breakpoint (;P) commands allow you to set and proceed from breakpoints at a vector instruction.

Note that, because the names of the vector registers (V0 through V15) and vector control registers (VCR, VLR, and VMR) are not defined in the DELTA/XDELTA symbol table, you cannot display their values using DELTA/XDELTA.

B.2.3.5.4 Vector Processing Support Within the VMS Patch Utility

Enhancements to the VMS Patch Utility allow it to interpret and display vector instructions that are replaced or deposited in a VAX MACRO program image file.

When issuing a REPLACE/INSTRUCTION instruction, you must supply the vector opcode, switches, and operands in the form and order defined by the VAX MACRO assembler notation. When displaying the contents of an image in instruction format, the Patch Utility produces vector instructions in this format. However, its hexadecimal listings present vector instructions in the instruction stream format defined by the VAX architecture; that is, an opcode followed by the vector control word in immediate addressing mode. (See the *VAX MACRO and Instruction Set Reference Manual* for descriptions of the assembler notation and instruction stream formats of vector instructions.)

B.2.3.6 Servicing Vector Exceptions

During the execution of an image, the image can incur a fatal error known as an exception condition. If the image has not declared a condition handler, the system forces the image to exit and displays a message indicating the reason for the exception. If the image has declared a condition handler, VMS transfers control to the handler to manage the exception. (See the *Introduction to VMS System Services* for a description of how to write and declare a condition handler.)

There are two major classes of vector processing exceptions:

- Memory management exceptions, including access violations, vector alignment faults, and vector instruction references to I/O space
- Vector arithmetic exceptions

VMS reports exceptions in the first category (memory management exceptions) as forms of access violation, using the signals SS\$_ACCVIO and SS\$_VECALIGN (see Table B-4). The exception argument list VMS supplies when signaling vector memory management exceptions is identical to the one it supplies with scalar access violations, except that VMS defines two additional bits in the reason mask to indicate the nature of the vector exception: a vector operation on an improperly aligned vector element in memory (bit 3) and vector instruction reference to an I/O space address (bit 4).

VMS reports exceptions in the second category (vector arithmetic exceptions) using the signal SS\$_VARITH (see Table B-4). As defined by the VAX vector architecture (see the *VAX MACRO and Instruction Set Reference Manual*), vector operate instructions always execute to completion. If an exception occurs, the default result is written as follows:

- The low-order 32 bits of the true result for integer overflow.
- Zero for floating underflow if exceptions are disabled.

- An encoded reserved operand for floating divide by zero, floating overflow, reserved operand, and enabled floating underflow. For vector convert instructions that convert floating-point data to integer data, where the source element is a reserved operand, the value written to the destination element is UNPREDICTABLE.

Table B-4 provides a summary of the means by which VMS signals vector processing exceptions and the arguments it provides for condition handlers. For information about how these exception conditions are reported by the VMS message facility, see Section B.2.2.7.

Table B-4 Summary of Exception Conditions

Exception	Type	Description	Arguments												
SS\$_ACCVIO	Fault	Access violation	Two, as follows: <div><div><div>1. Reason for access violation. This is a mask with the following format:</div><table><tr><th>Bit</th><th>Description</th></tr><tr><td>0</td><td>Type of access violation: Clear if page table entry protection code did not permit intended access Set if P0LR, P1LR, or SLR length violation</td></tr><tr><td>1</td><td>Page table entry reference: Clear if specified virtual address is not accessible Set if associated page table entry is not accessible</td></tr><tr><td>2</td><td>Intended access: Clear if read Set if modify</td></tr><tr><td>3</td><td>Vector alignment exception: Set if vector element is not properly aligned in memory¹</td></tr><tr><td>4</td><td>Vector instruction reference of I/O space Set if vector instruction referred to an I/O space address</td></tr></table></div><div><div>2. Virtual address to which access was attempted or, on some processors, virtual address within the page to which access was attempted. For access violations that occur due to a vector alignment exception or a vector instruction reference to I/O space, this virtual address is <i>always</i> an address within the page to which access was attempted.</div></div></div>	Bit	Description	0	Type of access violation: Clear if page table entry protection code did not permit intended access Set if P0LR, P1LR, or SLR length violation	1	Page table entry reference: Clear if specified virtual address is not accessible Set if associated page table entry is not accessible	2	Intended access: Clear if read Set if modify	3	Vector alignment exception: Set if vector element is not properly aligned in memory ¹	4	Vector instruction reference of I/O space Set if vector instruction referred to an I/O space address
Bit	Description														
0	Type of access violation: Clear if page table entry protection code did not permit intended access Set if P0LR, P1LR, or SLR length violation														
1	Page table entry reference: Clear if specified virtual address is not accessible Set if associated page table entry is not accessible														
2	Intended access: Clear if read Set if modify														
3	Vector alignment exception: Set if vector element is not properly aligned in memory ¹														
4	Vector instruction reference of I/O space Set if vector instruction referred to an I/O space address														

¹ Note that the VMS operating system reports this exception with an SS\$_VECALIGN fault.

(continued on next page)

VMS Version 5.4 Features

Servicing Vector Exceptions

Table B-4 (Cont.) Summary of Exception Conditions

Exception	Type	Description	Arguments												
SS\$_ILLVECOP	Fault	Illegal vector opcode. ²	Four, as follows: <ol style="list-style-type: none">1. Signal name, SS\$_ILLVECOP2. Illegal opcode that caused the exception3. Program counter (PC) of the vector instruction that caused the exception to be reported. (Note that this instruction is not always the one that caused the exception.)4. Processor status longword (PSL) at the time the exception is reported.												
SS\$_VARITH	Trap	Vector arithmetic trap	Five, as follows: <ol style="list-style-type: none">1. Signal name, SS\$_VARITH.2. Exception summary. This is a mask, the bits of which, when set, signify the following:<table><tr><th>Bit</th><th>Meaning</th></tr><tr><td>0</td><td>Floating underflow</td></tr><tr><td>1</td><td>Floating divide by zero</td></tr><tr><td>2</td><td>Floating reserved operand</td></tr><tr><td>3</td><td>Floating overflow</td></tr><tr><td>5</td><td>Integer overflow</td></tr></table>3. Vector register mask, the bits of which (0 through 15) correspond to the VAX vector registers (V0 through V15). When set, a bit indicates that an element of the associated vector register was involved in an operation that caused one or more of the vector arithmetic exceptions reported in the exception summary argument.4. Program counter (PC) of the vector instruction that caused the exception to be reported. (Note that this instruction is not always the one that caused the exception.)5. Processor status longword (PSL) at the time the exception is reported.	Bit	Meaning	0	Floating underflow	1	Floating divide by zero	2	Floating reserved operand	3	Floating overflow	5	Integer overflow
Bit	Meaning														
0	Floating underflow														
1	Floating divide by zero														
2	Floating reserved operand														
3	Floating overflow														
5	Integer overflow														
SS\$_VECALIGN	Fault	Vector alignment exception	Identical to the argument list for SS\$_ACCVIO												

² Note that some processors report illegal vector opcodes with the SS\$_OPCDEC exception.

(continued on next page)

Table B-4 (Cont.) Summary of Exception Conditions

Exception	Type	Description	Arguments
SS\$_VECDIS	Fault	Vector processor disabled	<p>Three, as follows:</p> <ol style="list-style-type: none"> Reason for vector disabled exception. The reason argument can have any of the following values: <ul style="list-style-type: none"> SS\$_NOPRIV—An ACL on the vector capability has denied a user-mode program access to the vector processor. SS\$_MCHECK—The vector processor has been disabled due to the detection of a hardware error. SS\$_INSFMEM—Insufficient nonpaged dynamic memory exists to turn the current process into a vector consumer. SS\$_CPUNOTACT—The VAX system contains no vector-present processor on which to schedule the current process. SS\$_BADCONTEXT—The vector state of the mainline routine is corrupt and cannot be restored. SS\$_EXQUOTA—The VMS operating system cannot allocate sufficient space to save the vector state of the mainline routine because the process in which the routine is executing has exceeded process paging file quota. SS\$_INSFWSL—The VMS operating system cannot allocate sufficient space to save the vector state of the mainline routine because the working-set limit of the process in which the routine is executing is too low. SS\$_VASFUL—The VMS operating system cannot allocate sufficient space to save the vector state of the mainline routine because the address space (P0 space) of the process in which the routine is executing is full. Program counter (PC) of the vector instruction that caused the exception to be reported. (Note that this instruction is not always the one that caused the exception.) Processor status longword (PSL) at the time the exception is reported.

B.2.3.7 Requirements of the VAX Procedure Calling and Condition Handling Standard

This section contains excerpts from the VAX Procedure Calling Standard that describe the requirements that procedures must follow when using the system's vector processing resources.

Code generated by VAX vectorizing compilers adheres to the rules described in this section. VAX MACRO code containing vector instructions must be written to comply with these requirements.

B.2.3.7.1 Vector Register Usage The VAX Calling Standard specifies no conventions for preserved vector registers, vector argument registers, or vector function value return registers. All such conventions are by agreement between the calling and called procedures. In the absence of such an agreement, all vector registers, including V0 through V15, VLR, VCR, and VMR are scratch registers. Among cooperating procedures, a procedure that does preserve or otherwise manipulate the vector registers by agreement with its callers must provide an exception handler to restore them during an unwind.

B.2.3.7.2 Vector and Scalar Processor Synchronization There are two kinds of synchronization between a scalar and vector processor pair: memory synchronization and exception synchronization. Sections B.2.3.7.3 and B.2.3.7.4 describe these types of synchronization.

B.2.3.7.3 Memory Synchronization Every procedure is responsible for synchronization of memory operations with the calling procedure and with procedures it calls. If a procedure executes vector loads or stores, the following must occur:

- An MSYNC instruction (a form of the MFVP instruction) must be executed before the first vector load/store to synchronize with memory operations issued by the caller. While an MSYNC instruction might typically occur in the entry code sequence of a procedure, exact placement can also depend on a variety of optimization considerations.
- An MSYNC instruction must be executed after the last vector load/store to synchronize with memory operations issued after return. While an MSYNC instruction might typically occur in the return code sequence of a procedure, exact placement can also depend on a variety of optimization considerations.
- An MSYNC must be executed between each vector load/store and each standard call to other procedures to synchronize with memory operations issued by those procedures.

That is, any procedure that executes vector loads or stores is responsible for synchronizing with potentially conflicting memory operations in any other procedure. However, execution of an MSYNC to ensure scalar/vector memory synchronization can be omitted when it can be determined for the current procedure that all possibly incomplete vector load/stores operate only on memory that is not accessed by other procedures.

B.2.3.7.4 Exception Synchronization Every procedure is responsible for ensuring that no exception can be raised after the current frame is changed (as a result of either a CALL or RET). If a procedure executes any vector instruction that might possibly raise an exception, then a SYNC instruction (a form of the MFVP instruction) must be executed prior to any subsequent CALL or RET.

However, if it can be determined that the only possible exceptions that can occur are ensured to be reported by an MSYNC instruction that is otherwise needed for memory synchronization, then the SYNC is redundant and can be omitted as an optimization.

Moreover, if it can be determined that the only possible exceptions that can occur are ensured to be reported by one or more MFVP instructions that read the vector control registers, then the SYNC is redundant and can be omitted as an optimization.

B.2.3.7.5 Synchronization Summary Memory synchronization with the caller of a procedure that uses the vector processor is required because there might be scalar machine writes (to main memory) still pending at the time of entry to the called procedure. The various forms of write-cache strategies allowed by the VAX architecture combined with the possibly independent scalar and vector memory access paths imply that a scalar store followed by a CALL followed by a vector load is not safe without an intervening MSYNC.

Within a procedure that uses the vector processor, proper memory and exception synchronization might require use of an MSYNC instruction or a SYNC instruction, or both, prior to calling another procedure or upon being called by another procedure. Further, for calls to other procedures, the requirements may vary from call to call depending on details of actual vector usage.

An MSYNC instruction (without SYNC) at procedure entry, procedure exit, and prior to a call, should provide proper synchronization in most cases. A SYNC instruction (without an MSYNC prior to a CALL or RET) will sometimes be appropriate. The remaining two cases, where both or neither MSYNC and SYNC are needed, are probably rare.

Refer to the *VAX MACRO and Instruction Set Reference Manual* for the specific rules on what exceptions are ensured to be reported by MSYNC and other MFVP instructions.

B.2.3.7.6 Condition Handler Parameters and Invocation If the VAX vector hardware or emulator option is in use, then, for hardware-detected exceptions, the vector state is implicitly saved before any condition handler is entered and restored after the condition handler returns. (No save/restore is required for exceptions that are initiated by calls to LIB\$SIGNAL or LIB\$STOP because there can be no useful vector state at the time of such calls in accordance with the rules given for vector register usage in Section B.2.3.7.1.) A condition handler can thus make use of the system vector facilities in the same manner as mainline code.

The saved vector state is not directly available to a condition handler. A condition handler that needs to manipulate the vector state to carry out agreements with its callers can call the \$RESTORE_VP_STATE service. This service restores the saved state to the vector registers (whether hardware or emulated) and cancels any subsequent restore. The vector state can then be manipulated directly in the normal manner by means of vector instructions. (This service is normally of interest only during processing for an unwind condition.)

B.2.3.8 VMS Accounting Utility (ACCOUNTING) Resource Packet Format

The VMS Accounting Utility uses the longword field ACR\$L_VP_CPUTIME in the resource packet (ACR\$K_RESOURCE) to record the vector CPU time (measured in 10-millisecond clock ticks) accrued by a process or image.

See the *VMS Accounting Utility Manual* for a complete description of the format and contents of ACCOUNTING records.

B.2.3.9 VMS Monitor Utility (MONITOR) VECTOR Class Record

As discussed in *VMS Monitor Utility Manual*, the VMS Monitor Utility (MONITOR) writes binary performance data to a VMS RMS sequential file known as the MONITOR recording file. Once per recording interval, MONITOR writes to this file a record containing data pertinent to each currently selected class. Version 5.4 of the VMS Monitor Utility includes the VECTOR class record, which contains data describing the time during which vector consumers have been scheduled on a vector-present processor.

See Section B.12.3 for a complete description of the MONITOR VECTOR command and the VECTOR class. See Section B.12.4 for specific information about the VECTOR class record and format.

B.3 Introduction to DECdtm Services

The VMS Version 5.4 operating system includes DECdtm services, which provide system services that demarcate and coordinate distributed transactions. By using the two-phase commit protocol, these services ensure consistent execution of distributed transaction on the VMS operating system. In turn, these system services make use of underlying logging and communication primitives necessary to enable distributed transaction commitment.

This section describes how the DECdtm services coordinate distributed transaction processing. The following sources in this manual also describe aspects of VMS Version 5.4 support for DECdtm services:

- Section B.11 (Log Manager Control Program Utility (LMCP))
- Section B.12.1 (MONITOR TRANSACTION command and TRANSACTION class)
- Section B.19 (of this manual) and the *VAX RMS Journaling Manual* (RMS Journaling support)
- *VMS Version 5.4 Release Notes*

Note

By default, processes for DECdtm services are started when a full VMS boot is executed. Before any transactions can be started, however, you must first use the Log Manager Control Program Utility (LMCP) to create a transaction log file (as described in Section B.11).

If you do *not* want to run DECdtm software, you can prevent the startup of DECdtm processes by defining the systemwide logical name SYS\$DECDTM_INHIBIT in the SYS\$MANAGER:SYLOGICALS.COM command procedure. You can define SYS\$DECDTM_INHIBIT to be any string. For example:

```
$ DEFINE/SYSTEM/EXEC SYS$DECDTM_INHIBIT "yes"
```

See the *Guide to Setting Up a VMS System* for more information about the SYLOGICALS.COM command procedure.

B.3.1 Characteristics of Distributed Transactions

In business terminology, a transaction is a discrete unit of work. One example of a transaction is the purchasing of tickets from an airline reservation system. Another example is the transferring of funds between customer accounts using an automated teller machine (ATM). In both examples, the processing of the transaction involves interaction with databases.

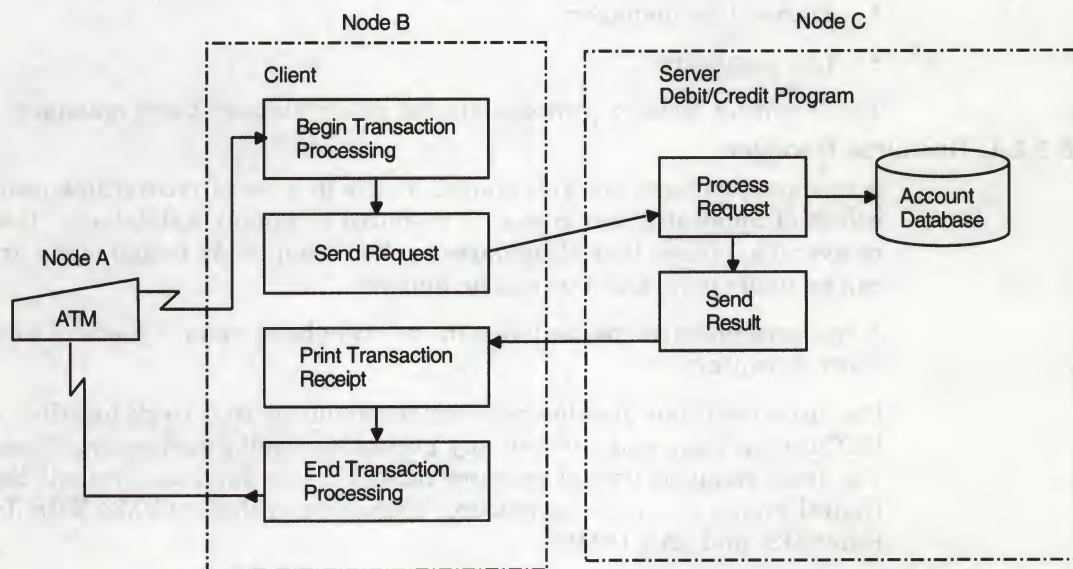
Characteristically, transaction processing incorporates large, corporate-level applications that support many users for critical business functions. In transaction processing applications, there are usually many users simultaneously performing predefined functions (query and update) to a collection of shared data, generally a database. Results are usually expected immediately.

Another characteristic of transaction processing is that it is usually distributed. Transaction execution typically involves communication between a client program and one or more databases that can be locally or remotely located. This communication between client and server might typically take place through a network of systems distributed at various geographic locations; hence, the operation can be called distributed transaction processing. In the example of funds transfers at an ATM, the central system—or database—acts as a server, providing services to the customer—or client—at the ATM.

A single transaction represents the execution of a set of procedures. A client and the server must communicate using read and write operations to enable the client program to perform the desired task; for example, to perform a debit/credit operation to transfer funds in customer accounts.

Figure B-4 shows the execution flow of a simple debit/credit application. A user at the ATM requests a financial operation, such as a transfer of funds from one account to another. A client program on Node A receives this request from the ATM. The client program forwards the request to a debit/credit program on Node B, and the debit/credit program updates the customer accounts database. The transaction shown in this figure is distributed because the cooperating programs are located on different computer systems.

Figure B-4 Sample Debit/Credit Transaction Execution



ZK-1221A-GE

For transaction processing to be reliable, every required operation involved in the execution of the transaction must be completed before the transaction is made permanent; otherwise, none of the operations are completed. A transaction that has this characteristic, known as atomicity, is considered an **atomic** transaction.

An atomic transaction must execute in its entirety or must have no effect at all. A transaction that executes in its entirety is called committed. One that terminates prematurely (and therefore has no effect) is called aborted.

The DECdtm services implement a commit protocol to guarantee atomic transaction processing. This protocol, known as the two-phase commit protocol, ensures atomicity by sequencing the commit process in such a way as to ensure that all resources (for example, databases) will be committed.

In the funds transfer example, it is vital that each of the customer's accounts is properly debited or credited and the account files updated only after it has been acknowledged that the transfer has occurred. If a system failure occurs while the transaction is processing, all of the previous operations of the transaction must be nullified. This arrangement keeps the database consistent; no operation is ever partially applied to the database.

B.3.2 Transaction Processing System Model

In Digital's model for transaction processing, several components work together to execute atomic transactions.

At the end-user level, user-written application programs define the task to be accomplished, such as query, update, and debit/credit. Application programs also specify how transactions are to be executed. The application programs initiate transaction execution using calls to VMS system services.

At the system level, the execution of the transaction depends on the interaction of the three main transaction processing components:

- Resource managers
- Transaction managers
- Log managers

The following sections provide detailed descriptions of these managers.

B.3.2.1 Resource Manager

A resource manager controls shared access to a set of recoverable resources on behalf of application programs. A resource is usually a database. The term recoverable means that all updates to the resources on behalf of the transaction can be made permanent or can be undone.

A resource manager participates in the two-phase commit protocol to commit or abort a transaction.

Resource managers provide recovery mechanisms that work together with the DECdtm services and perform any necessary logging and recovery operations. The most common type of resource manager is a database system. Several Digital products can act as resource managers, including VMS RMS Journaling, Rdb/VMS, and VAX DBMS.

The execution of a transaction can span several nodes. The root application program can use the services of one or more resource managers on its home node. An application can also communicate with applications on other nodes, and these remote applications can also use other resource managers.

B.3.2.2 Transaction Manager

A transaction manager supports the services issued from application programs to start, end, and abort transactions. A transaction manager coordinates the action of a distributed transaction by sending instructions to resource managers about how to complete the transaction.

In a distributed network of transaction processing systems, each VMS node normally contains one DECdtm object. This object contains the transaction manager for transactions initiated from that node. The transaction manager maintains a list of participants in a transaction. In the execution of a transaction, participants may include:

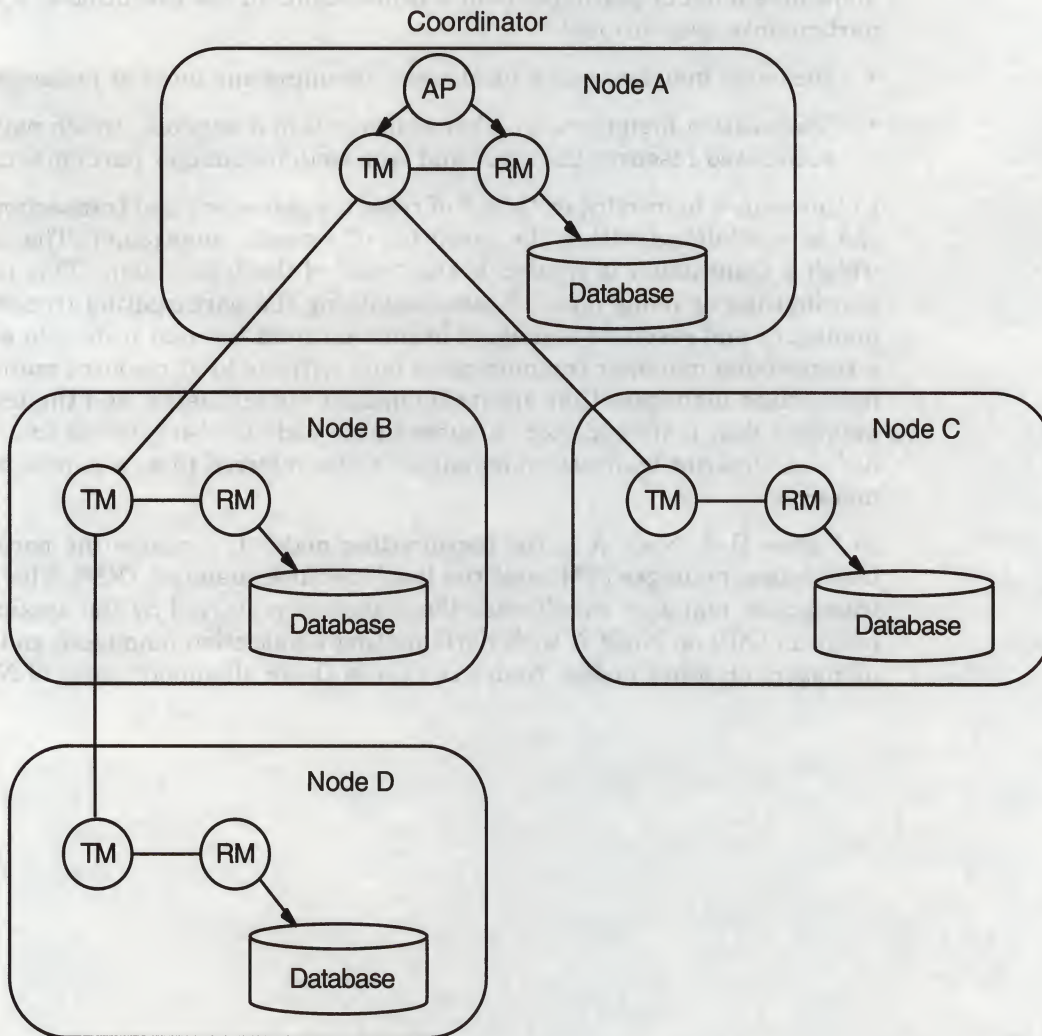
- Resource managers on a local node, spanning one more or processes
- Transaction managers on other nodes within a network, which may also have associated resource manager and transaction manager participants

In this way, a hierarchy, or "tree," of resource managers and transaction managers can be established within the execution of a single transaction. The node on which a transaction is created is the "root" of the transaction. This is the coordinating or home node. Nodes containing the participating transaction managers and resource managers branch off from the root node. On each node, a transaction manager communicates only with its local resource managers, the transaction managers that are its immediate subordinates, and the transaction manager that is its superior. A subordinate node is also referred to as a child node. A superior transaction manager is also referred to as a parent transaction manager.

In Figure B-5, Node A is the coordinating node. It contains the parent transaction manager (TM) and the local resource manager (RM). The parent transaction manager coordinates the transaction started by the application program (AP) on Node A with participating transaction managers and resource managers on other nodes. Nodes B, C, and D are all subordinates of Node A.



Figure B-5 Participants in a Distributed Transaction Example



ZK-1870A-GE

B.3.2.3 Log Manager

A log manager provides the mechanism for storing a permanent record of the execution of distributed transactions in log files. Each recoverable resource manager implements its own log manager component, which consists of a set of logging services. Logging services are also provided by the DECdtm services. During normal operation, resource managers and transaction managers write log files containing records of transaction state information. After recovering from a failure, a resource manager or transaction manager can read the log file to determine the state of a transaction at the time of failure.

B.3.3 Overview of Two-Phase Commit Protocol

Specific transaction management system services called from application programs mark the start and end of a transaction. The DECdtm system services include:

- Start Transaction (\$START_TRANS)
- Start Transaction and Wait (\$START_TRANSW)
- End Transaction (\$END_TRANS)
- End Transaction and Wait (\$END_TRANSW)
- Abort Transaction (\$ABORT_TRANS)
- Abort Transaction and Wait (\$ABORT_TRANSW)

The transaction manager component of the DECdtm services coordinates the execution of these system services. See the *VMS System Services Reference Manual* for more detailed descriptions of the DECdtm system services new for Version 5.4 of the VMS operating system.

The processing of a distributed transaction begins when an application calls the \$START_TRANS or \$START_TRANSW service. In response, the transaction manager generates a unique transaction identifier (TID) for the transaction so that it can keep track of the transaction. The transaction manager uses the TID to identify all actions performed by resource managers and transaction managers on behalf of the transaction.

Each resource manager is responsible for providing recovery capabilities for its own resources by performing transaction logging. The transaction manager is responsible for notifying all resource managers involved in a transaction of all relevant transaction-state transitions. The transaction manager keeps track of the state of each transaction in case a system or process fails before the transaction completes.

The transaction manager maintains a list of resource managers and transaction managers that participate in a transaction's execution. The transaction manager uses this list of participants to execute the two-phase commit protocol. During the execution of this protocol, each participating transaction manager writes transaction information to a log file. A log file contains a permanent record of transaction states. By having access to a log file, a transaction manager can resume the execution of the two-phase commit protocol after recovering from a system failure.

For a complete description of transaction log files, see Section B.11.

Each participating resource manager supports atomic transactions on its resources. To do this, the resource manager notifies the transaction manager as soon as that resource manager is first accessed by the application. A resource manager logs enough information to allow it to undo or redo operations it performed on behalf of a transaction. Similar to a transaction manager, a resource manager logs transaction state changes to a log file.

The processing of a transaction completes when one of the following calls is made:

- Commit—Using \$END_TRANS or \$END_TRANSW
- Planned abort—Using \$ABORT_TRANS or \$ABORT_TRANSW

VMS Version 5.4 Features

Overview of Two-Phase Commit Protocol

(See the *VMS System Services Reference Manual* for more detailed descriptions of the DECdtm system services introduced in Version 5.4 of the VMS operating system.)

Upon receiving an End Transaction call, the DECdtm services implement the two-phase commit protocol to inform all participants how to proceed with the execution of the transaction.

The first phase of the two-phase commit protocol is the prepare phase. During this phase, the transaction manager uses a polling mechanism to determine if the participants can complete all the steps involved in a given transaction and can therefore commit the transaction. A participant that has successfully prepared casts a "yes" vote. If an error occurs during the polling that prevents a participant from responding—for example, if a resource manager fails or if a network link goes down—a "no" vote is assumed.

A "yes" vote indicates that the participating resource manager can either commit or abort the operations performed within this transaction, even if a failure occurs.

If all of the participants declare that they can commit by voting "yes," the transaction manager makes a decision to commit and proceeds to the second phase, known as the commit phase.

The transaction manager now orders the participants to commit the transaction. At this point all participants complete their transaction operations.

If any of the participants fails to prepare successfully, the transaction is aborted. The transaction manager orders all remaining participants to abort the transaction and roll back their transaction processing work. Thus, none of the actions of the distributed transaction are made permanent.

B.3.4 Managing DECdtm Services Using VMS Utilities

The VMS operating system provides the following utilities to manage the information provided by the DECdtm services:

- The Log Manager Control Program Utility (LMCP) is used to create and manage log files that are used by transaction managers. See Section B.11 for a complete description.
- The VMS Monitor Utility can be used to monitor the status of transactions executing on the system. See Section B.12 for more information.

B.3.5 New TRANSACTION_ID Data Type for Programming Routines

To support DECdtm programming routines, there is a new VMS data type, or structure, for low- and high-level languages. The **transaction_id** data type is an octaword that stores a unique transaction identifier.

B.4 VMS Version 5.4 General User Features

This section describes enhancements to the following components of the VMS operating system:

- DCL Commands
- System Messages
- DECwindows User and Desktop Applications

B.4.1 DCL Commands

Table B-5 contains a summary of DCL commands that are new or enhanced but are not being printed for VMS Version 5.4.

See the command section following the table for details of the BACKUP/MEDIA_FORMAT qualifier and the MOUNT/MEDIA_FORMAT qualifier. Refer to the revised *VMS DCL Dictionary* for complete descriptions of the remaining new and enhanced VMS Version 5.4 DCL commands and lexical functions.

Table B-5 Summary of New and Enhanced DCL Commands

Command	Enhancements
BACKUP	Now includes new /MEDIA_FORMAT qualifier, which controls data compaction on tape drives that support data compaction.
FONT	New command that compiles fonts for use by the DECwindows server and converts an ASCII bitmap distribution format (BDF) into binary server natural form (SNF).
MOUNT	Now includes new /MEDIA_FORMAT qualifier, which controls data compaction on tape drives that support data compaction.
PSWRAP	New command that invokes the PSWRAP translator, which converts procedures written in PostScript to callable routines.
SHOW ZONE	New command that displays the current state of a VAXft 3000 system.
START/ZONE	New command that adds a zone to a running VAXft 3000 system.
STOP/ZONE	New command that removes a zone from a running VAXft 3000 system.
VIEW	Now accepts new PS input format, which lets you use the CDA Viewer to view PostScript files (which use the file extension .PS).

BACKUP/MEDIA_FORMAT=[NO]COMPACTION

Output Save-Set Qualifier

Controls whether data records are automatically compacted and blocked together. Data compaction and record blocking increase the amount of data that can be stored on tape drives that support data compaction.

The compaction ratio depends on the data and the tape drive you use. For more information, see the documentation supplied with your tape drive.

Format

input-specifier output-save-set-spec /MEDIA_FORMAT=[NO]COMPACTION

Description

The /MEDIA_FORMAT qualifier can be used only with tape drives that support data compaction.

Once data compaction has been selected for a tape cartridge, compaction is used for the entire cartridge until you initialize the cartridge with the /MEDIA_FORMAT=NOCOMPACTION qualifier.

Example

```
$ BACKUP WORK$:[TESTFILES...]*.*;* MUA0:TEST.SAV -  
_$/MEDIA_FORMAT=COMPACTION /REWIND
```

This command saves all files in the directory [TESTFILES] and its subdirectories in a save set named TEST.SAV. The /MEDIA_FORMAT=COMPACTION qualifier specifies that the tape drive automatically compacts and blocks together data records on the tape.

MOUNT/MEDIA_FORMAT=[NO]COMPACTION

Enables and controls data compaction and record blocking on tape drives that support data compaction.

Format

/MEDIA_FORMAT=[NO]COMPACTION device-name

Description

The /MEDIA_FORMAT qualifier allows you to mount a tape cartridge and enable data compaction and record blocking on tape drives that support data compaction. Data compaction and record blocking increase the amount of data that can be stored on a single tape cartridge.

Records can either be compacted and blocked, or they can be recorded in the same way that they would be recorded by a noncompaction drive. Note that for compacting tape drives, once data compaction or noncompaction has been selected for a given cartridge, that status applies to the entire cartridge.

Additionally, when you enable data compaction, caching is automatically enabled.

Example

```
$ MOUNT/MEDIA_FORMAT=COMPACTION MUA0: BOOKS
```

In this example, a tape device labeled BOOKS is mounted with data compaction and record blocking enabled.

B.4.2 System Messages

This section provides information about installing and accessing an online help version of the *VMS System Messages and Recovery Procedures Reference Manual*.

B.4.2.1 System Messages Available from Online Help

With Version 5.4 of the VMS operating system, you can now install and access an optional online help version of the *VMS System Messages and Recovery Procedures Reference Manual*. Because this is a large file, it is *not* included as part of the default root library, SYS\$HELP:HELPLIB.HLB. You can access the file, named SYS\$HELP:SYMSGHELP.HLB, as follows:

- Use the /LIBRARY qualifier with the HELP command. For example:

```
$ HELP/LIBRARY=SYS$HELP:SYMSGHELP.HLB ERRORS ACCVIO
```
- Define a logical name that instructs the help system to search the new help library when it does not find the specified topic in the VMS root help library. For example:

```
$ DEFINE HLP$LIBRARY DISK$2:[QUAIL]SYMSGHELP  
$ HELP ERRORS DISMAL
```

In this example, the DEFINE statement creates a logical name for the help library that the help system is to search after it has searched the root library, SYS\$HELP:HELPLIB.HLB.

The help system first searches the root library for ERRORS. When it does not find an error,¹ it then searches the library defined by HLP\$LIBRARY until it finds ERRORS and displays the appropriate information. For information about defining logical names and search patterns for the help system, see the HELP COMMAND in the *VMS DCL Dictionary*.

- Using the VMS Librarian Utility, you can extract the ERRORS module from SYMSGHELP.HLB and insert it into the default root help library, HELPLIB.HLB. This allows direct access without using extra HELP qualifiers or logical names. For more information, see the *VMS Librarian Utility Manual*.

The system messages help library is in compressed format. Decompressing the library gives you faster access to it but requires an additional 1600 blocks of disk space. To decompress the library, enter a command similar to the following:

```
$ LIBRARY/DATA=EXPAND/OUTPUT=SYS$SYSROOT:[SYSHLP]SYMSGHELP.HLB -  
_ $ SYS$SYSROOT:[SYSHLP]SYMSGHELP.HLB
```

In this example, SYS\$SYSROOT is the name of the device where the file is located and [SYSHLP] is the name of the directory.

Note

The system messages help library is not decompressed when you execute the LIBDECOMP.COM procedure described in the *VMS Version 5.4 Upgrade and Installation Manual*.

You can use the VMS tailoring utility (VMSTAILOR) to add or delete the system messages help library. Deleting this library does not affect the other help libraries.

¹ Previous versions of HELPLIB.HLB provided information about the format of system messages under the name ERROR. This information is now named FORMAT_OF_ERROR.

B.4.3 DECwindows User and Desktop Applications

This section describes new features of interest to DECwindows users. These features include enhancements to the Session Manager, the CDA Viewer, Calculator, Clock, and Mail.

B.4.3.1 Session Manager

Enhancements to the Session Manager include the addition of new languages to the Customize Language dialog box and the ability to change your target screen, as described in Section B.4.3.2 and Section B.4.3.3 respectively.

B.4.3.2 Setting Another Session Language

The following languages have been added to the Customize Language dialog box in the Session Manager:

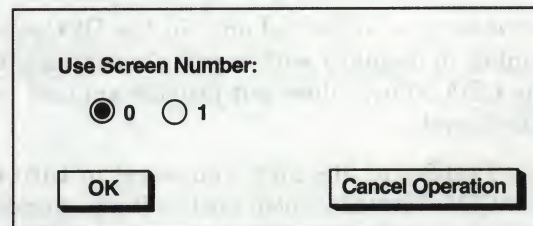
- Australian
- Austrian
- Belgian Dutch
- Belgian French
- Danish
- Fiji
- Finnish
- Hebrew
- New Zealand
- Papua New Guinea
- Portuguese

For more information about setting another session language, see the Version 5.3 edition of the *VMS DECwindows User's Guide*.

B.4.3.3 Changing Your Target Screen

When you run an application or choose Print Screen on a workstation that supports more than one screen display, by default DECwindows displays a dialog box asking you which screen you want to use (see Figure B-6).

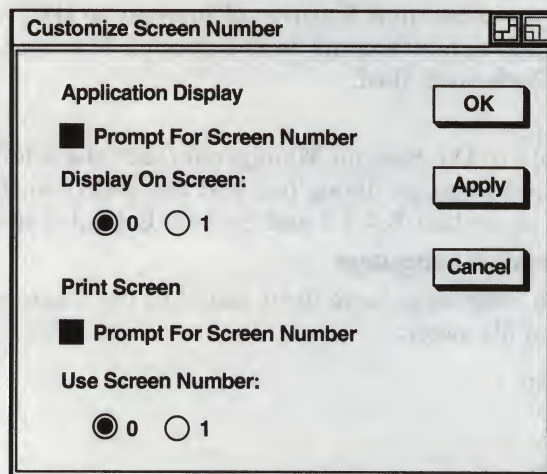
Figure B-6 DECwindows Screen Number Dialog Box



ZK-1959A-GE

If you want to use the same screen every time you run an application or use PrintScreen, you can disable the screen number prompt and choose your target screen. To disable the screen number prompt or change your target screen, choose Screen Number... from the Session Manager's Customize menu. The Session Manager displays the Customize Screen Number dialog box (see Figure B-7).

Figure B-7 DECwindows Customize Screen Number Dialog Box



ZK-1958A-GE

When you choose your target screen in the Customize Screen Number dialog box, DECwindows will run applications (or PrintScreen) on the screen you designated. If you click on the Prompt For Screen buttons, DECwindows will not display the screen number dialog box.

B.4.3.4 CDA Viewer

The DECwindows CDA Viewer now lets you view PostScript files. Section B.4.3.5 describes how to view a PostScript file and Section B.4.3.6 describes the new processing options available.

B.4.3.5 Viewing a PostScript File

To view a PostScript file, select the CDA Viewer menu item from the FileView Applications menu. In the Open window, click on PS in the File Format box and then select the PostScript file you want to view.

From a DCL window, enter the VIEW command in the following format to open a PostScript document for viewing:

```
VIEW filename.PS /FORMAT=PS /INTERFACE=DECWINDOWS
```

When you invoke the CDA Viewer from the DCL prompt, you do not need to specify processing options for the PostScript files.

PostScript file viewing is supported only in the DECwindows CDA Viewer and only when running to displays with servers containing the Display PostScript Extension. The CDA Viewer does not provide support for PostScript files on character-cell terminals.

When viewing a PostScript file, after you select or turn to a particular page, you can click on the CDA Viewer Cancel button if you decide not to view the page while it is being processed. The CDA Viewer immediately stops processing that page.

B.4.3.6 New Processing Options for Viewing PostScript Files

In addition to the Default Paper Size option, new processing options specific to viewing PostScript files are available in the Paper Size dialog box. The additional PostScript options are highlighted, unless you already chose PS as the file format to display.

These options are valid only for viewing PostScript files and are ignored for all other file formats:

- Orientation radio box

The Orientation radio box lets you select the orientation for displaying PostScript files. By default, the CDA Viewer displays files in the same portrait or landscape mode in which they were created. You can use the Orientation radio box to select different orientations to view files in reverse landscape mode or upside down.

- Scale Factor option

The Scale Factor option lets you scale the page display size of your PostScript file. The number you select indicates whether the CDA Viewer will shrink or enlarge the page display. If the scale factor is less than 1.0, the page display will shrink. If the number is greater than 1.0, the page display will expand. You can specify a scale factor in the range of 0.1 to 4.0 times the size of the original page display. By default, a typical page display has a scale factor of 1.0.

- Use Comments toggle button

The Use Comments option specifies that the CDA Viewer should interpret file-structure comments that often appear in PostScript files. This enables the CDA Viewer to detect the location of page breaks in a PostScript file, for example.

The Use Comments option is enabled by default. This is indicated by the highlighted Use Comments toggle button.

You can disable the Use Comments option by clicking on it before opening your PostScript file. This is recommended in instances where the PostScript file contains comments that are not correct, causing the CDA Viewer to either display the PostScript file incorrectly or generate an error message. In most cases, disabling the Use Comments option and reopening the file corrects the problem.

- Use Bitmap Widths toggle button

The Use Bitmap Widths option adjusts the display of your PostScript file for improved viewing on the screen. By default, a printed PostScript file has a finer resolution, or more dots per inch, than a PostScript file displayed on a screen. If you try to view the printed format of a PostScript file on line, the page layout will be the same, but the text may be dense and difficult to read.

To clarify your PostScript file for online viewing, you can specify the Use Bitmap Widths option so that the CDA Viewer will use spacing formulas designed for bitmaps (screen images) instead of those designed for print.

The Use Bitmap Widths option is disabled by default. If you select the Use Bitmap Widths option, the next time you open a PostScript file, the CDA Viewer will use bitmap widths to display your file. Text characters will appear well spaced and easy to read. However, the file may look slightly different on screen than it would when printed. Columns may not be aligned

precisely or a paragraph formatted for right justification may appear instead with a ragged right margin.

- Use Fake Trays toggle button

The Use Fake Trays option lets you view a PostScript file that contains tray size directives. Tray size directives are instructions that tell the printer what paper tray size to use. These directives, however, are specific to certain printers (such as the LPS40) and are not part of the Display PostScript language.

By default, the CDA Viewer ignores tray size directives if you try to display a PostScript file that contains them. To override that default behavior and view tray size directives in a PostScript file (to identify occurrences of nonstandard PostScript, for example), click on the Use Fake Trays option and reopen the file.

- Watch Progress toggle button

The Watch Progress option lets you view a PostScript file while it is being processed for display in the CDA Viewer window. You can view a page as it is being processed, rather than waiting to view the entire page after it has been processed.

B.4.4 Calculator

Calculator now has two additional modes: hexadecimal and octal. When you first start the Calculator, it is in decimal mode. A new Mode menu contains Hexadecimal and Octal menu entries for changing modes. The keyboard display and functions change according to the mode.

B.4.5 Clock

Clock now has a menu bar with File, Customize, and Help menus for interacting with Clock. The menu bar provides an alternative to the previous method of pressing MB2 while pointing to the Clock display.

The only menu item under File is Quit. Choose Quit to exit from Clock.

The Customize menu lets you change the Clock display. The Customize menu has three menu items. The menu items correspond to the Settings..., Save Settings, and Use System Settings previously available on a pop-up menu. Choosing the Settings... menu item displays the Clock Settings dialog box. The only change to the dialog box is the addition of a toggle button for Menu Bar. By default, the Menu Bar button is shaded and the menu bar is displayed. If you do not want the menu bar displayed, click on the Menu Bar button.

Help is now available directly as a menu on the menu bar, rather than from a pop-up dialog box.

B.4.6 Mail: Displaying PostScript Files

Mail can now display PostScript files, provided the files you send or receive contain *only* PostScript language. A PostScript file always begins with a percent sign and an exclamation point (%!). If any other text precedes the %!, Mail cannot display the file. For example, when mail is forwarded, additional text (in the form of extra mail headers) is often inserted at the beginning of the file. Because this additional text precedes the %!, Mail cannot display the PostScript file correctly. To avoid this problem, use an editor to remove all headers before you forward a mail message in PostScript format. Similarly, if you receive a PostScript file that does not display properly, use an editor to remove all headers (or any other

text that precedes the %!) and forward the file to yourself. The file should then display properly.

B.5 VMS Version 5.4 System Management Features

This section describes enhancements to the following components of the VMS operating system:

- Autogen Command Procedure
- VAXcluster Management
- Error Log Utility (ERROR LOG)
- System Security
- Log Manager Control Program Utility (LMCP)
- Monitor Utility (MONITOR)
- Network Control Program Utility (NCP)
- VMS Volume Shadowing Phase II

B.6 AUTOGEN Command Procedure

This section describes changes to the AUTOGEN command procedure in Version 5.4 of the VMS operating system.

B.6.1 Parameter Name Validation

When AUTOGEN reads a parameter file such as MODPARAMS.DAT, it now checks to determine if the parameter names specified in the file are valid. If a parameter name is invalid, a warning message is written to AGEN\$PARAMS.REPORT (a new file described further in Section B.6.2). The following is an example of this warning message:

```
** WARNING ** - Invalid parameter name: LPRCOUNT
               The following record is suspect:
               LPRCOUNT = 34
```

AUTOGEN checks only the parameter name. It does not check the validity of the value specified for the parameter.

If a parameter name is invalid, the line is *not* ignored. AUTOGEN attempts to use the specified value.

A parameter name is not checked if it is specified in a line that contains a DCL expression other than the symbol assignment (=). For example, AUTOGEN does not check the validity of a parameter name specified in a line with a DCL IF statement. Instead, AUTOGEN writes a warning message to AGEN\$PARAMS.REPORT. The following is an example of this message:

```
** WARNING ** - DCL command detected
               Parameter validation turned off for:
               IF WINDOW_SYSTEM = 1 THEN NPAGEDYN = 250000
```


B.6.2 AGEN\$FEEDBACK.REPORT Replaced by New File

The file SYS\$SYSTEM:AGEN\$FEEDBACK.REPORT has been replaced by a new file called SYS\$SYSTEM:AGEN\$PARAMS.REPORT. This new file includes all of the information previously contained in AGEN\$FEEDBACK.REPORT, as well as information about the non-feedback parameters and additional messages. Many of the warning and informational messages that AUTOGEN previously displayed on the screen are now written to AGEN\$PARAMS.REPORT.

For example, when AUTOGEN finds multiple MIN_, MAX_, or ADD_ values for a single parameter, AUTOGEN writes a warning message to AGEN\$PARAMS.REPORT. The warning message includes the parameter name, the value being used for the MIN_, MAX_, or ADD_ value, and the value being superseded. The following are examples of this type of message:

```
** WARNING ** - Multiple ADD records for ADD_LRPCOUNT found.
                  VMS value (300) combining with MODPARAMS value (400)
                  Value used is 700

** WARNING ** - Multiple MIN values found for MIN_LRPCOUNTV.
                  Using VMS value (1000) which is superseding MODPARAMS value (800)

** WARNING ** - Multiple MAX values found for MAX_SWAPFILE2_SIZE.
                  Using MODPARAMS value (1000) which is superseding VMS value (1200)
```

When AUTOGEN uses feedback information to calculate the value for a new parameter, this information is written to AGEN\$PARAMS.REPORT. The following is an example of this type of message:

```
MAXPROCESSCNT parameter information:
Feedback information.
    Old value was 41. New value is 50
    Maximum Observed Processes: 35
```

When an AUTOGEN calculation is overridden by a value specified in a parameter file, AUTOGEN writes a message to AGEN\$PARAMS.REPORT. This message includes the new parameter value and the reason why the parameter was overridden. AUTOGEN will write this message for any parameter value that overrides AUTOGEN's calculations, whether the value is supplied by the system manager or by Digital. The following is an example of this type of message:

```
LONGWAIT parameter information:
Override Information - parameter calculation has been overridden.
    The calculated value was 30. The new value is 10.
    LONGWAIT has been disabled by a hard-coded value of 10.
```

B.6.3 MODPARAMS.DAT Includes External Parameter Files

To aid in cluster management, AUTOGEN can now read external parameter files specified within MODPARAMS.DAT. This feature allows system managers to maintain both clusterwide and system-specific versions of AUTOGEN parameters.

To include a parameter file, place the following command in MODPARAMS.DAT or in any subsequent parameter file:

```
AGEN$INCLUDE_PARAMS full-directory-specification:filename
```

Note

If an include statement is the first line in MODPARAMS.DAT, AUTOGEN attempts to resolve all subsequent parameter settings. For example, if AUTOGEN finds two MIN_ statements for the same parameter, it uses

the higher value. If the statements cannot be resolved, AUTOGEN uses the parameter setting specified after the include file.

The following is an example of a MODPARAMS.DAT that includes an external parameter file:

```
! include system wide parameter settings
!
AGEN$INCLUDE_PARAMS SYS$COMMON:[SYSMGR]COMMON_CI_NODE_MODPARAMS.DAT
MIN_LRPCOUNT = 45
DUMPSTYLE = 0
.
.
.
```

This example reads the parameter file named `SYS$COMMON:[SYSMGR]COMMON_CI_NODE_MODPARAMS.DAT` before reading the parameters specified after the include statement in MODPARAMS.DAT. If the included file in this example specified the parameter setting `DUMPSTYLE = 1`, AUTOGEN would override this setting with the statement `DUMPSTYLE = 0`, which is specified after the include statement in MODPARAMS.DAT.

The format of all included parameter files should be the same as MODPARAMS.DAT. For information about MODPARAMS.DAT, see the description of AUTOGEN in the *Guide to Setting Up a VMS System*.

B.6.4 MIN_, MAX_, and ADD_ Values Allowed for Page and Swap Files

You can now control the size of page and swap files by specifying MIN_, MAX_, and ADD_ values in a parameter file. The syntax for specifying MIN_, MAX_, and ADD_ values is identical to that used with other parameters.

For example, you can control the size of general page and swap files by including one or more of the following lines in a parameter file:

```
PAGEFILE = 20000
ADD_PAGEFILE = 5000
MIN_SWAPFILE = 1500
MAX_SWAPFILE = 4000
```

You can also specify the sizes of individual page and swap files (including secondary files) by including one or more of the following lines in a parameter file:

```
SWAPFILE1_SIZE = 2000
ADD_PAGEFILE1_SIZE = 2000
MIN_PAGEFILE2_SIZE = 3000
MAX_SWAPFILE3_SIZE = 3000
```

Note

You cannot specify a MIN_, MAX_, or ADD_ value for both a general page or swap file and a specific page or swap file.

B.6.5 New Feedback Parameters

The existing parameters LRPCOUNT and LNMSHASHTBL are now feedback parameters. This means that AUTOGEN can set these parameters using data collected in AUTOGEN feedback mode. You should remove any values for LRPCOUNT and LNMSHASHTBL that are specified in MODPARAMS.DAT, including MIN_, MAX_ and ADD_ values, so that AUTOGEN can set these parameters using feedback information.

B.6.6 Logical Names Defined by AUTOGEN

To aid in system management, AUTOGEN defines three process logical names to indicate how AUTOGEN was last run. These logical names are assigned a character string value each time AUTOGEN is run on a system. The following table lists and describes the logical names:

Logical Name	Description
AGEN\$P1	The starting phase of AUTOGEN, for example, SAVPARAMS.
AGEN\$P2	The end phase of AUTOGEN, for example, TESTFILES. If an error occurred that caused AUTOGEN to abort, then "_E" is appended to the phase name, for example, GENPARAMS_E.
AGEN\$P3	The mode of execution, that is, either FEEDBACK or NOFEEDBACK.

B.6.7 New Technique for Running AUTOGEN in Batch Mode

As of Version 5.2-1 of the VMS operating system, Digital recommends a new technique for running AUTOGEN. This technique automates AUTOGEN feedback, allowing the system manager to receive reports from multiple systems on a regular basis. To use this technique, create a batch-oriented procedure that runs AUTOGEN in two stages. A sample command procedure is shown in Example B-1.

The first stage of the command procedure runs AUTOGEN at peak times to collect data on realistic system loads. The following command accomplishes this task:

```
$ @SYS$UPDATE:AUTOGEN SAVPARAMS SAVPARAMS FEEDBACK
```

Executing this command does not affect the performance of the system.

The second stage of the command procedure runs AUTOGEN again during off-peak hours to interpret the data collected in the first stage. The following command accomplishes this task:

```
$ @SYS$UPDATE:AUTOGEN GETDATA TESTFILES FEEDBACK
```

The procedure sends the resulting report, contained in the file AGEN\$PARAMS.REPORT, to the SYSTEM account using the following MAIL command:

```
$ MAIL/SUBJECT="AUTOGEN FEEDBACK REPORT FOR system-name" -  
  SYS$SYSTEM:AGEN$PARAMS.REPORT SYSTEM
```

Review this report on a regular basis to see whether the load on a system has changed. If AUTOGEN's calculations are different from the current values, correct the tuning by executing AUTOGEN with one of two commands:

- If the system can be shut down and rebooted immediately, execute the following command:

```
$ @SYS$UPDATE:AUTOGEN GETDATA REBOOT FEEDBACK
```


VMS Version 5.4 Features New Technique for Running AUTOGEN in Batch Mode

- If the system cannot be shut down and rebooted immediately, execute the following command to reset the system parameters:

```
$ @SYS$UPDATE:AUTOGEN GETDATA SETPARAMS FEEDBACK
```

The new parameters will take effect the next time the system boots.

The sample command procedure shown in Example B-1 will run AUTOGEN in the new technique described. Use this procedure only as an example; create a similar command procedure as necessary to meet your requirements.

Example B-1 Sample AUTOGEN Command Procedure

```
$ BEGIN$: ! ++++++ AGEN_BATCH.COM ++++++
$ on warning then goto error$
$ on error then goto error$
$ on severe_error then goto error$
$ on control_y then goto error$
$!
$! Setup process
$!
$! Set process information
$ set process/priv=all/name="AUTOGEN Batch"
$! Keep log files to a reasonable amount
$ purge/keep=5 AGEN_Batch.log
$ time = f$time() ! Fetch current time
$ hour = f$integer(f$cvtime(time,, "hour")) ! Get hour
$ today = f$cvtime(time,, "WEEKDAY") ! Get Day of the week
$ if f$integer(f$cvtime(time,, "minute")) .ge. 30 then hour = hour + 1
$!
$! Start of working day...
$!
$ 1AM$:
$ if hour .le. 2
$ then
$ next_time = "today+0-14"
$ gosub submit$ ! Resubmit yourself
$ set noon
$!
$! Run AUTOGEN to setparams using the parameter values collected earlier
$! in the day (i.e., yesterday at 2:00pm)
$ if today .eqs. "Tuesday" .OR. today .eqs. "Thursday" .OR. -
today .eqs. "Saturday"
$ then
$ @sys$update:autozen getdata testfiles feedback
$ mail/sub="Autogen Feedback Report for system-name" -
sys$system:agen$params.report system
$! Clean up
$ purge/keep=7 sys$system:agen$feedback.report
$ purge/keep=7 sys$system:agen$feedback.dat
$ purge/keep=7 sys$system:params.dat
$ purge/keep=7 sys$system:autozen.par
$ purge/keep=7 sys$system:setparams.dat
$ purge/keep=7 sys$system:agen$addhistory.tmp
$ purge/keep=7 sys$system:agen$addhistory.dat
$ endif
$ goto end$
$ endif
$!
```

(continued on next page)

Example B-1 (Cont.) Sample AUTOGEN Command Procedure

```

$ 2PM$:
$ if hour .le. 15
$ then
$   next_time = "today+0-17"
$   gosub submit$
$   if today .eqs. "Monday" .OR. today .eqs. "Wednesday" .OR. -
today .eqs. "Friday"
$   then
$     @sys$update:autozen savparams savparams feedback
$   endif
$   goto end$
$ endif
$!
$ 5PM$:
$ if hour .le. 18
$ then
$   next_time = "tomorrow+0-1"
$   gosub submit$
$   endif
$!
$! End of working day...
$!
$ END$:      ! ----- BATCH.COM -----
$ exit
$!++
$! Subroutines
$!--
$!
$ SUBMIT$:
$ submit/name="AGEN_Batch"/restart/noprint -
$ /log=AGEN_batch.log -
$ /queue=sys$batch/after="'"next_time'" sys$system:AGEN_batch.com
$ return
$!++
$! Error handler
$!--
$ ERROR$:
$ mail/sub="AGEN_BATCH.COM - Procedure failed." _nl: system
$ goto end$

```

B.6.8 Using MAIL to Send AGEN\$PARAMS.REPORT

After closing the AGEN\$PARAMS.REPORT file, AUTOGEN now checks for the existence of a file named SYS\$UPDATE:AGEN\$MAIL.COM. If this file exists, it is executed from within AUTOGEN. (Note, however, that AUTOGEN does not execute AGEN\$MAIL.COM during VMS upgrades or installations or after minimum system boots.)

You can use AGEN\$MAIL.COM alone or with the batch-oriented procedure described in Section B.6.7 to send AGEN\$PARAMS.REPORT to the SYSTEM account or to an account of your choice. To do so, create a command procedure named SYS\$UPDATE:AGEN\$MAIL.COM that includes the following command:

```

$ MAIL/SUBJECT="AUTOGEN FEEDBACK REPORT FOR system-name" -
  SYS$SYSTEM:AGEN$PARAMS.REPORT SYSTEM

```

If you use the AGEN\$MAIL.COM procedure along with the batch-oriented procedure described in Section B.6.7, AGEN\$MAIL.COM replaces the MAIL command line in the batch-oriented command procedure.

B.7 VAXcluster Management

This chapter describes enhancements to the following VAXcluster components:

- Computer interconnect (CI) architecture extensions
- Mass storage control protocol (MSCP) server load sharing
- Preferred path support for Digital Storage Architecture (DSA) disks

See the revised *VMS VAXcluster Manual* for more information.

B.7.1 CI Architecture Extensions

Extensions to the computer interconnect (CI) architecture allow the application of multiple CI interfaces per CPU and multiple star couplers per VAXcluster system. These extensions make possible VAXcluster systems with many times the data-throughput capacity of current VAXcluster systems with a single star coupler.

B.7.2 MSCP Server Load Sharing

Beginning with Version 5.4 of the VMS operating system, mass storage control protocol (MSCP) servers monitor their I/O traffic and periodically calculate a Load Available rating to indicate available capacity for I/O requests.

Load Available is calculated by counting the read and write requests sent to the server and periodically converting this to requests per second and subtracting this calculated value from the server's Load Capacity (also specified in requests per second).

This information is communicated to the VMS Version 5.4 MSCP class drivers (DUDRIVER and DSDRIVER). When a disk is mounted or a failover occurs, the class driver selects the server with the highest Load Available rating to access the disk.

Load Balancing is enabled and controlled by the SYSGEN parameters MSCP_LOAD and MSCP_SERVE_ALL. In most cases, the values established by CLUSTER_CONFIG.COM are appropriate.

MSCP_SERVE_ALL determines whether the server participates in load balancing. If it is set to 2 (serve only local disks), the server does not monitor its I/O traffic and does not participate in load balancing. Other valid settings for MSCP_SERVE_ALL (0, 1) result in the server monitoring I/O traffic and communicating Load Available information to the class drivers.

MSCP_LOAD is used to communicate Load Capacity to the server, in addition to its existing function of controlling the loading of the MSCP server. If it is set to 1, the MSCP server is loaded and its Load Capacity is set to a default value based upon CPU type. If MSCP_LOAD is set to a value greater than 1, the server is loaded and its Load Capacity set to that value.

As before, setting MSCP_LOAD to zero disables loading of the MSCP server.

B.7.3 Preferred Path Support for DSA disks

The VMS Version 5.4 operating system lets you specify a preferred path for Digital Storage Architecture (DSA) disks. This includes RA series disks and disks accessed through the MSCP server.

If a preferred path is specified for a disk, the MSCP disk class drivers (DUDRIVER and DSDRIVER) use the path as their first attempt to locate the disk and bring it on line as a result of a DCL MOUNT command or failover of an already mounted disk.

VMS Version 5.4 Features

Preferred Path Support for DSA disks

In addition, it is possible to initiate failover of a mounted disk to force the disk to the preferred path or to use load balancing information for disks accessed via MSCP servers.

The preferred path is specified by a \$QIO function, IO\$_SETPRFPTH, with the P1 parameter containing the address of a counted ASCII string (.ASCIC). This string is the node name of the HSC or VMS system that is to be the preferred path. The node name must match an existing node known to the local node and, if it is a VMS system, it must be running the MSCP server. This function does not move the disk to the preferred path. For more information about the IO\$_SETPRFPTH function, refer to the *VMS I/O User's Reference Manual: Part I*.

B.8 System Generation Utility (SYSGEN)

This section describes enhancements to the VMS System Generation Utility (SYSGEN) that are new for Version 5.4 of the VMS operating system.

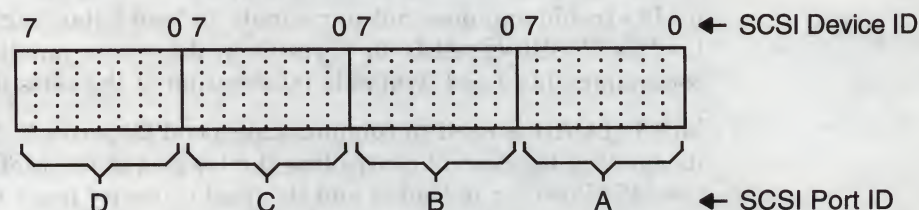
B.8.1 SCSI_NOAUTO Parameter

The VMS Version 5.4 operating system defines the special SYSGEN parameter SCSI_NOAUTO for use with MicroVAX or VAXstation configurations that include third-party Small Computer System Interface (SCSI) devices. (See the *VMS Device Support Manual* for more information about SCSI devices.) The SYSGEN parameter SCSI_NOAUTO replaces the SYSGEN parameter VMUSD1.

SYSGEN's autoconfiguration facility automatically loads the VMS SCSI disk or tape class driver for a device on the SCSI bus that identifies itself as either a random-access or sequential-access device. If this SCSI device is to be supported instead by the VMS generic SCSI class driver or a third-party SCSI class driver, the automatic loading of a VMS SCSI class driver for the device must be disabled.

The SCSI_NOAUTO parameter, as shown in Figure B-8, allows a configuration including a SCSI third-party device to prevent the loading of a VMS disk or tape SCSI class driver for any given device ID.

Figure B-8 SCSI_NOAUTO System Parameter



ZK-1371A-GE

The SCSI_NOAUTO system parameter stores a bit mask of 32 bits, where the low-order byte corresponds to the first SCSI bus (PKA0), the second byte corresponds to the second SCSI bus (PKB0), and so on. For each SCSI bus, setting the low-order bit inhibits automatic configuration of the device with SCSI device ID 0; setting the second low-order bit inhibits automatic configuration of the device with SCSI device ID 1, and so forth. For instance, the value 00002000₁₆ would prevent the device with SCSI ID 5 on the bus identified by

SCSI port ID *B* from being configured. By default, all of the bits in the mask are cleared, allowing all devices to be configured.

B.8.2 LOAD_PWD_POLICY Parameter

The SYSGEN parameter LOAD_PWD_POLICY works in conjunction with the Set Password Utility and with LOGINOUT (if you are forced to change your password at login). This parameter controls whether or not the Set Password Utility or LOGINOUT attempts to use site-specific password policy routines, which are contained in the shareable image SYS\$LIBRARY:VMS\$PASSWORD_POLICY.EXE. The default is 0.

Installing and enabling a site-specific password policy image requires both SYSPRV and CMKRNL privileges. To set the LOAD_PWD_POLICY parameter, enter the following commands:

```
$ RUN SYS$SYSTEM:SYSGEN
SYSGEN> USE ACTIVE
%%%%%%%%
SYSGEN> SET LOAD_PWD_POLICY 1
SYSGEN> WRITE ACTIVE
SYSGEN> WRITE CURRENT
```

To make the changes permanent, modify the system parameter file, MODPARAMS.DAT, so the parameter LOAD_PWD_POLICY is set to 1.

For descriptions of site-defined password filters for the VMS Version 5.4 operating system, see Section B.10 and the *VMS System Services Reference Manual*.

B.8.3 LOAD_SYS_IMAGES Parameter

The LOAD_SYS_IMAGES parameter controls the loading of system images described in the system image data file, VMS\$SYSTEM_IMAGES.DATA. Currently, you can replace three system services with services specific to your site:

- \$ERAPAT—Generates a security erase pattern
- \$MTACCESS—Controls magnetic tape access
- \$HASH_PASSWORD—Applies a hash algorithm to an ASCII password

The *VMS System Services Reference Manual* describes how to create a system service image and how to copy the image into the SYS\$LOADABLE_IMAGES directory and add an entry for it in the VMS system images file using the SYSMAN utility. After generating a new system image data file, you reboot the system to load in your service.

If you have difficulty booting with the site-specific system services and therefore do not want the site-specific system services loaded, you can set the parameter of LOAD_SYS_IMAGES to 0 during SYSBOOT. The default is 1.

B.8.4 Supported Device Names for VAXft 3000 Systems

With Version 5.4 of the VMS operating system, the System Generation Utility (SYSGEN) supports the following device types in VAXft 3000 systems:

Code Name	Device Type
CM	Environmental control monitor
GD	DMA driver

VMS Version 5.4 Features

Supported Device Names for VAXft 3000 Systems

Code Name	Device Type
EF	Logical Ethernet driver
EP	Physical Ethernet driver
PW	DSSI disk driver
SF	Logical DSF driver
SM	Physical DSF driver

B.8.5 New SYSGEN Commands

This section describes the following new SYSGEN commands:

- SHOW/BI=BIindex
- SHOW/BUS=busId
- SHOW/XMI=BIindex

SHOW/BI=Blindex

The SHOW/BI=Blindex command displays device addresses that are currently mapped in the I/O space for the VAXBI bus. It also displays node and nexus numbers and generic names of UNIBUS and MASSBUS adapters, VAXBI adapters, memory controllers, and interconnection devices such as the DR32 and CI.

Use of the SHOW/BI=Blindex command requires the CMEXEC privilege.

Format

SHOW/BI=Blindex

Example

SYSGEN> SHOW/BI

(CPU Type: VAX 8800

Cpu Connection: NMI

```

** Bus map for BI 00 on 28-FEB-1990 14:13:02.95 **
Address 20000000 (node 00) responds with value 0108 CI
Address 20004000 (node 02) responds with value 0106 BI - NMI Adapter (NBIB)
Address 2000E000 (node 07) responds with value 0109 BI Combo Board (DMB32)
** Bus map for BI 01 on 28-FEB-1990 14:13:03.00 **
Address 22000000 (node 00) responds with value 0102 UB
Address 22004000 (node 02) responds with value 0106 BI - NMI Adapter (NBIB)
Address 2200E000 (node 07) responds with value 410F BI - NI Adapter (DEBNA))

```

The command in this example displays device addresses that are currently mapped in the I/O space for the BI bus and additional information about the BI bus adapters.

SHOW/BUS=busId

The SHOW/BUS=busId command displays the buses and any subsequent attached buses and all attached device node numbers, generic names of processors, memory modules, adapters, VAXBI adapters, memory controllers, and interconnection devices such as the NI.

Use of the SHOW/BUS command requires the CMEXEC privilege.

Format

SHOW/BUS=busId

Example

SYSGEN> SHOW/BUS

Cpu Type: VAX 8800			Cpu Connection: NMI	
Bus	Node	Generic Name	Nexus(hex)	Connection Address
BI 00	00	CI	0000	
BI 00	02	BI - NMI Adapter (NBIB)	0002	
BI 0	07	BI Combo Board (DMB32)	0007	
BI 01	00	UB	0010	
BI 01	02	BI - NMI Adapter (NBIB)	0012	
BI 01	07	BI - NI Adapter (DEBNA)	0017	

The command in this example displays information about all the adapters on the system buses.

SHOW/XMI=Blindex

The SHOW/XMI=Blindex command displays device addresses that are currently mapped in the I/O space for the XMI bus. It also displays node and nexus numbers and generic names of processors, adapters, VAXBI adapters, memory controllers, and interconnection devices such as the NI.

Use of the SHOW/XMI=Blindex command requires the CMEXEC privilege.

Format

SHOW/XMI=Blindex

Example

SYSGEN> SHOW/XMI

```
** Bus map for XMI 00 on 28-FEB-1990 14:14:50.48 **
Address 21880000 (node 01) responds with value 8082 XMI - 6000-400 processor
Address 21900000 (node 02) responds with value 8082 XMI - 6000-400 processor
Address 21980000 (node 03) responds with value 8082 XMI - 6000-400 processor
Address 21A00000 (node 04) responds with value 8082 XMI - 6000-400 processor
Address 21A80000 (node 05) responds with value 8082 XMI - 6000-400 processor
Address 21B00000 (node 06) responds with value 4001 XMI - memory module
Address 21B80000 (node 07) responds with value 4001 XMI - memory module
Address 21C00000 (node 08) responds with value 4001 XMI - memory module
Address 21C80000 (node 09) responds with value 4001 XMI - memory module
Address 21D00000 (node 0A) responds with value 4001 XMI - memory module
Address 21D80000 (node 0B) responds with value 4001 XMI - memory module
Address 21E00000 (node 0C) responds with value 0C03 XMI - NI adapter (DEMNA)
Address 21E80000 (node 0D) responds with value 2001 XMI - BI Adapter (DWMB/A)
Address 21F00000 (node 0E) responds with value 2001 XMI - BI Adapter (DWMB/A)
```

The command in this example displays device addresses that are currently mapped in the I/O space for the XMI bus and additional information about the XMI bus adapters.

VMS Version 5.4 Features

B.9 Error Log Utility (ERROR LOG)

B.9 Error Log Utility (ERROR LOG)

This section describes enhancements to the VMS Error Log Utility (ERROR LOG) that are new for Version 5.4 of the VMS operating system.

B.9.1 Supported Device Types for VAXft 3000 Systems

With Version 5.4 of the VMS operating system, the Error Log Utility supports the following device types in VAXft 3000 systems:

Code Name	Device Type
CM	Environmental control monitor
DSF32	Synchronous communications adapter
GD	DMA driver
EF	Logical Ethernet driver
EP	Physical Ethernet driver
PW	DSSI disk driver
RF31	DSSI fixed hard disk
SF	Logical DSF driver
SM	Physical DSF driver
TF70	DSSI magnetic tape drive

B.9.2 New Keywords for /EXCLUDE and /INCLUDE Qualifiers

The /EXCLUDE and /INCLUDE qualifiers accept new device-class and entry-type keywords, described in the following table:

Device-Class Keyword	Function
ADAPTER	Includes or excludes entries for adapter errors
CACHE	Includes or excludes entries for memory caching errors
INFORMATIONAL	Includes or excludes error log entries such as media quality reports from magnetic tape devices
VECTOR	Includes or excludes entries for vector processing errors

Entry-Type Keyword	Function
CONFIGURATION	Includes or excludes entries that describe system configuration
SYNDROME	Includes or excludes VAX 9000 console-generated entries that provide encoded syndrome values used by Customer Services

B.9.3 New Qualifier: /NODE

The Error Log Utility now accepts the /NODE qualifier. See the following command description for more information.

ERROR LOG/NODE

This qualifier enables you to generate a report consisting of error log entries for specific nodes in a VAXcluster system.

Format

/NODE =(node-name[,...])

Parameter

node-name

Specifies the names of one or more VAXcluster members. Names cannot exceed six characters. If more than one node name is entered, you must specify a comma-separated list of node names enclosed in parentheses.

Example

```
$ ANALYZE/ERROR_LOG/NODE=(ORANGE,NASSAU) ERRLOG.OLD;72
```

In this example, a VAXcluster includes members ORANGE, PUTNAM, and NASSAU. However, the output consists of only those entries that were logged for VAXcluster members ORANGE and NASSAU.

B.10 System Security

This section describes new features of the VMS Version 5.4 operating system that system managers can use to enhance the security of their systems by implementing various password features.

B.10.1 Site-Defined Password Policy

Starting with the VMS Version 5.4 operating system, passwords selected by users can be screened for acceptability. The VMS system automatically compares new passwords against a system dictionary to ensure that a password is not a native-language word. It also maintains a history list of a user's passwords and compares each new password against this list to guarantee that an old password is not reused. Sites can screen passwords further by developing and installing an image that filters passwords for words that are particularly sensitive to the installation.

In addition, a site with contractual obligations to use special algorithms for encrypting passwords will be able to use them.

This section describes these security enhancements.

B.10.1.1 Screening New Passwords

Sites that choose to let users select their own passwords rather than use the password generator can now screen user-selected passwords. As of Version 5.4, the VMS system automatically compares new passwords against a system dictionary, which is stored in SYS\$LIBRARY, to ensure that a password is not a native-language word. The VMS system also maintains a list of all the passwords a user has had during the year and compares each new password against this history list to guarantee that an old password is not reused.

Both the dictionary and the history search can be disabled through the Authorize Utility. You disable the dictionary search with the DISPWDDIC option to the /FLAGS qualifier; you disable the history search with the DISPWDHIS option to the /FLAGS qualifier.

B.10.1.1.1 Password History List VMS keeps a year's worth of data in the password history list. If the password limit is exceeded, the system forces a user to accept generated passwords. By default, the list stores 60 passwords. A security administrator can change the defaults for the length of time passwords are retained and the maximum number of passwords per user.

Using the DCL command DEFINE, you can change the defaults for the capacity and lifetime of the password history list. For example, to increase the capacity of the history list from 60 passwords to 100, you would add the following line to the command procedure SYLOGICALS.COM, which is located in SYS\$MANAGER:

```
$ DEFINE/SYSTEM/EXEC SYS$PASSWORD_HISTORY_LIMIT 100
```

There is a correspondence between the lifetime of a password history list and the number of passwords allowed on the list. For example, if you increase the password history lifetime to four years and your passwords expire every two weeks, you would need to increase the password history limit to at least 104 (4 years times 26 passwords a year). The password history lifetime and limit can be changed dynamically, but they should be consistent across all nodes on the cluster.

Sites using secondary passwords might need to double the password limit to account for the secondary password storage.

The password history list is located in SYS\$SYSTEM. The list can be redirected off the system disk using the logical name VMS\$PASSWORD_HISTORY. This logical name should also be defined using /SYSTEM/EXEC and placed in SYS\$MANAGER:SYLOGICALS.COM.

B.10.1.1.2 Site-Specific Filter Security administrators can develop a site-specific password filter to ensure that passwords are not words readily associated with their site, for example, product names or personnel names. A filter can also check for particular character variations.

To create a list of site-specific words, you write the source code, create a shareable image, install the image, and, finally, enable the policy by setting a SYSGEN parameter. See the *VMS System Services Reference Manual* for step-by-step instructions. Installing and enabling a site-specific password filter requires both SYSPRV and CMKRNL privileges. In addition, if INSTALL and SYSPRV file-access auditing are enabled, multiple security alarms are generated when the password filter image is installed and the required change to the SYSGEN parameter is noted on the operator console.

The shareable image contains two global routines that are called by the VMS Set Password Utility whenever a user changes a password.

Warning

The two global routines allow a security administrator to obtain both the proposed plaintext password and its equivalent quadword hash value. All security administrators should be aware of this feature because its subversion by a malicious privileged user will compromise your system's security.

Digital recommends that you place security alarm ACEs on the password filter image and its parent directory. See the *VMS System Services Reference Manual* for instructions.

B.10.1.2 Specifying a Password Algorithm

The VMS operating system protects passwords from disclosure through encryption. VMS algorithms transform passwords from plaintext strings into cipher text, which is then stored in the user authorization file (UAF). Whenever a password check is done, the check is based on the encrypted password, not the plaintext password. The system password is always encrypted with an algorithm known to the VMS system.

The /ALGORITHM qualifier in the Authorize Utility allows you to define which algorithm the VMS system should use to encrypt a user's password, both primary and secondary. Your choices are the current VMS algorithm or a site-specific algorithm. The syntax is as follows:

/ALGORITHM=keyword=type [=value]

Table B-6 lists all the keywords and types you can specify with the /ALGORITHM qualifier.

To assign the VMS password encryption algorithm for a user, you would enter the following command:

UAF> MODIFY HOBBIT/ALGORITHM=PRIMARY=VMS

VMS Version 5.4 Features

Specifying a Password Algorithm

If a site-specific algorithm is selected, you must give a value to identify the algorithm:

```
UAF> MODIFY HOBBIT/ALGORITHM=CURRENT=CUSTOMER=128
```

The *VMS System Services Reference Manual* provides directions for using a customer algorithm. You must create a site-specific \$HASH_PASSWORD in which you define an algorithm number. This number has to correspond with the number used in the AUTHORIZE command MODIFY/ALGORITHM.

Whenever a user is assigned a site-specific algorithm, the Authorize Utility reports this information in the display provided by the SHOW command.

Table B-6 Arguments to the /ALGORITHM Qualifier

Keyword	Function
BOTH	Set the algorithm for primary and secondary passwords.
CURRENT	Set the algorithm for the primary, secondary, both, or no passwords depending on account status. Current is the default value.
PRIMARY	Set the algorithm for the primary password only.
SECONDARY	Set the algorithm for the secondary password only.
Type	Definition
VMS	The algorithm used in the version of VMS that is running on your system.
CUSTOMER	A numeric value in the range 128-255 identifies a customer algorithm.

B.11 Log Manager Control Program Utility (LMCP)

The Log Manager Control Program Utility (LMCP) is a component of DECdtm services residing within the VMS Version 5.4 operating system. The log manager ensures that, as each transaction is processed, a record of each transaction state is recorded in a log file on disk.

The DECdtm transaction manager invokes the log manager to write these transaction records as necessary, ensuring that a consistent transaction outcome is achieved even in the event of a system failure. Writing log records is necessary for the consistent recovery of the transaction-specific data.

This section describes how a system manager can use the Log Manager Control Program Utility (LMCP) to create and manage transaction log files, and it provides a complete description of all the LMCP commands.

See Section B.3 for a complete overview of DECdtm services.

B.11.1 Managing Transaction Log Files

To optimize the execution of distributed transactions on your system, you need to consider a number of factors relating to transaction log files. This section discusses these factors, providing recommendations and guidelines in the following areas:

- Using the SYS\$JOURNAL logical name
- Where to place a transaction log file
- How VAXcluster failover works

- Determining the initial size required for a transaction log file
- Creating a transaction log file
- Resizing a transaction log file

Note

To use LMCP commands, you must have SYSPRV privilege. To use the LMCP command CONVERT, you must have CMKRNL privilege. It is assumed throughout this section that system managers or other individuals who have these privileges will be implementing the procedures described herein.

B.11.1.1 Defining SYS\$JOURNAL

The logical name SYS\$JOURNAL defines the directory location where DECdtm services expect to find log files. SYS\$JOURNAL is a system-table, executive-mode logical name, normally defined in the SYS\$STARTUP:SYLOGICALS.COM command procedure.

If SYS\$JOURNAL is not defined in SYS\$STARTUP:SYLOGICALS.COM, then a default logical name value is defined as SYS\$COMMON:[SYSEXE].

You can define SYS\$JOURNAL using the following command format:

```
DEFINE/SYSTEM/EXEC SYS$JOURNAL device:[directory]
```

The logical name SYS\$JOURNAL can be defined as a search list. For example, the following command defines a search list consisting of two directories.

```
$ DEFINE/SYSTEM/EXEC SYS$JOURNAL DISK1:[LOGFILES], DISK2:[LOGFILES]
```

This example shows DISK1:[LOGFILES] to be the primary, or local, directory that DECdtm services always search first. DISK2:[LOGFILES] is the secondary directory; DECdtm services search this directory after the directory DISK1:[LOGFILES] is searched. If you create a transaction log file using the LMCP CREATE command, then the log file is placed in the first directory, DISK1:[LOGFILES].

If a transaction log file is created on a different node using DISK2:[LOGFILES] as the primary—or local—directory and DISK1:[LOGFILES] as the secondary directory, then the search list should specify the local log file directory first. Thus, the following command defines a search list consisting of two directories, where DISK2:[LOGFILES] is the local directory and the first to be searched by DECdtm services:

```
$ DEFINE/SYSTEM/EXEC SYS$JOURNAL DISK2:[LOGFILES], DISK1:[LOGFILES]
```

If you create a transaction log file using the LMCP CREATE command, then the log file is placed in the first directory, DISK2:[LOGFILES].

B.11.1.2 Placing a Transaction Log File

Transactions cannot be started until you have created a transaction log file, using the LMCP CREATE command. But before you create a transaction log file, you should consider where to locate it for best performance on your system.

A log file can be placed on any file-structured device that is available to the processor. The following list includes possible alternate locations for log files, in the recommended order:

1. Shadowed nonsystem disk

VMS Version 5.4 Features

Placing a Transaction Log File

2. Nonsystem disk
3. Shadowed system disk
4. System disk

For increased performance, follow the general guidelines for installing a secondary page/swap file. Use a high-performance, HSC-based disk that has little activity.

You should also take into account the following considerations when locating a log file:

- Shadowed versus nonshadowed disk

Because a transaction log file is almost exclusively write-only during normal processing, a shadowed disk may be slower than a nonshadowed disk. However, a shadowed disk provides increased data availability in the event of media failure.

- Local versus cluster disk

Although a disk on a local node can provide higher performance, particularly in an NI-based VAXcluster system, if that VAXcluster member node fails, other nodes in the VAXcluster will not be able to access the failed node's disk. (See Section B.11.1.3.) Therefore, it is better if disks are mounted across the VAXcluster and correctly defined using the logical name SYS\$JOURNAL. That way, if a node fails, other nodes can still access the failed node's disk.

In a VAXcluster, log files should be placed on disks accessible to all members of the VAXcluster. This practice facilitates VAXcluster failover by making the log files on each VAXcluster member node available to other VAXcluster members.

B.11.1.3 VAXcluster Failover

VAXcluster failover is a mechanism that DECdtm services provide to enable VAXcluster nodes to perform recovery for a member node that has failed.

To make VAXcluster failover work, you need to correctly define SYS\$JOURNAL (as described in Section B.11.1.1) so that DECdtm services can locate all transaction log files in use in the VAXcluster.

VAXcluster failover occurs only within a VAXcluster environment and is completely automatic and transparent to applications and resource managers using DECdtm services. VAXcluster failover starts when a VAXcluster member node fails and holds information that surviving VAXcluster member nodes need to process their transactions.

When VAXcluster failover is initiated, recovery proceeds while the failed node is rebooting. This allows other nodes that need information from the failed node to resolve transactions. It also allows resource managers to release locks on database records without waiting for the failed node to reboot.

Normally, each VAXcluster member node is primarily responsible for accessing its own transaction log file. Any node that requires information from a log file it does not have open must send a request for that information to the VAXcluster node member that currently has the log file open—the node normally responsible for that log file.

During VAXcluster failover, the first requesting node that requires information from a failed node opens the failed node's transaction log file to perform recovery. This action lets recovery on the failed node's log file begin while the failed node is rebooting. Normally, transaction recovery on the log file completes before the

failed node has rebooted. Therefore, nodes that had their transactions blocked by the failure of the VAXcluster node have their transactions resolved before the failed node reboots. The surviving VAXcluster members proceed as if the failed node had already rebooted.

Once a VAXcluster member node has opened the log file of a failed node, all further requests from other VAXcluster member nodes are directed to the node that has opened the log file. Only one VAXcluster member node can access a failed node's log file at any one time. When the failed node has rebooted, it reacquires access to its log file and requests are passed to that rebooted VAXcluster node member once again.

B.11.1.4 Determining Transaction Log File Size

Use the LMCP CREATE command to create transaction log files. The /SIZE qualifier of this command specifies the size of the log file in blocks. By default, the file size is 4000 blocks. However, since performance of transaction processing applications depend on transaction logging, Digital recommends that you plan ahead when creating log files.

A number of factors must be considered when estimating transaction log file requirements. These factors include the rate of transactions executed per second and the duration of the transactions. As a quick way to estimate log file size, Digital recommends the following algorithm:

$$\text{Transaction start rate} * \text{Transaction duration} * 40 = \text{log file size in disk blocks}$$

You can use the MONITOR TRANSACTION command of the Monitor Utility to determine the start rate and duration for transactions already executing on your system. (See Section B.12.1 for more information about the MONITOR TRANSACTION command.)

For example, if the start rate is 5 transactions per second and the duration is 10 seconds, the calculation is:

$$5 * 10 * 40 = 2000 \text{ blocks}$$

The recommended file size for a log file in this example is 2000 blocks.

Due to a number of factors, file size requirements can vary widely from one system to the next. Therefore, the guidelines listed here for determining log file size can provide only very rough estimates. When planning for log files, it is recommended that you overestimate, rather than underestimate, the file size.

B.11.1.5 Creating Transaction Log Files

Transactions cannot be started until a transaction log file exists. By default, processes for DECdtm services are started when a full VMS boot is executed.¹ The DECdtm process TP_SERVER then checks for the existence of a transaction log file on the system and continues checking every 15 seconds for the existence of a transaction log file on the system so that recovery can occur automatically, even if a log file's disk is not available when the system first boots.

To create a log file, use the LMCP CREATE command. Before creating a log file, you should understand the recommendations for placing and sizing log files, as described in Section B.11.1.2 and Section B.11.1.4.

¹ If you do not want to run DECdtm software, you can prevent the startup of DECdtm processes by defining the systemwide logical name SYS\$DECDTM_INHIBIT. See the note at the beginning of Section B.3 for more information.

VMS Version 5.4 Features

Creating Transaction Log Files

A log file must be named with the file name `SYSTEM$node-name`, where *node-name* is the name of the node on which the log file will be used. For example, a log file created on node ORANGE should be given the file name `SYSTEM$ORANGE`. The default file type is `LM$JOURNAL`.

The default file specification for the log file is:

```
SYS$JOURNAL: .LM$JOURNAL
```

B.11.1.6 Example of Creating a Transaction Log File

This section summarizes the steps involved in creating transaction log files for a sample VAXcluster system.

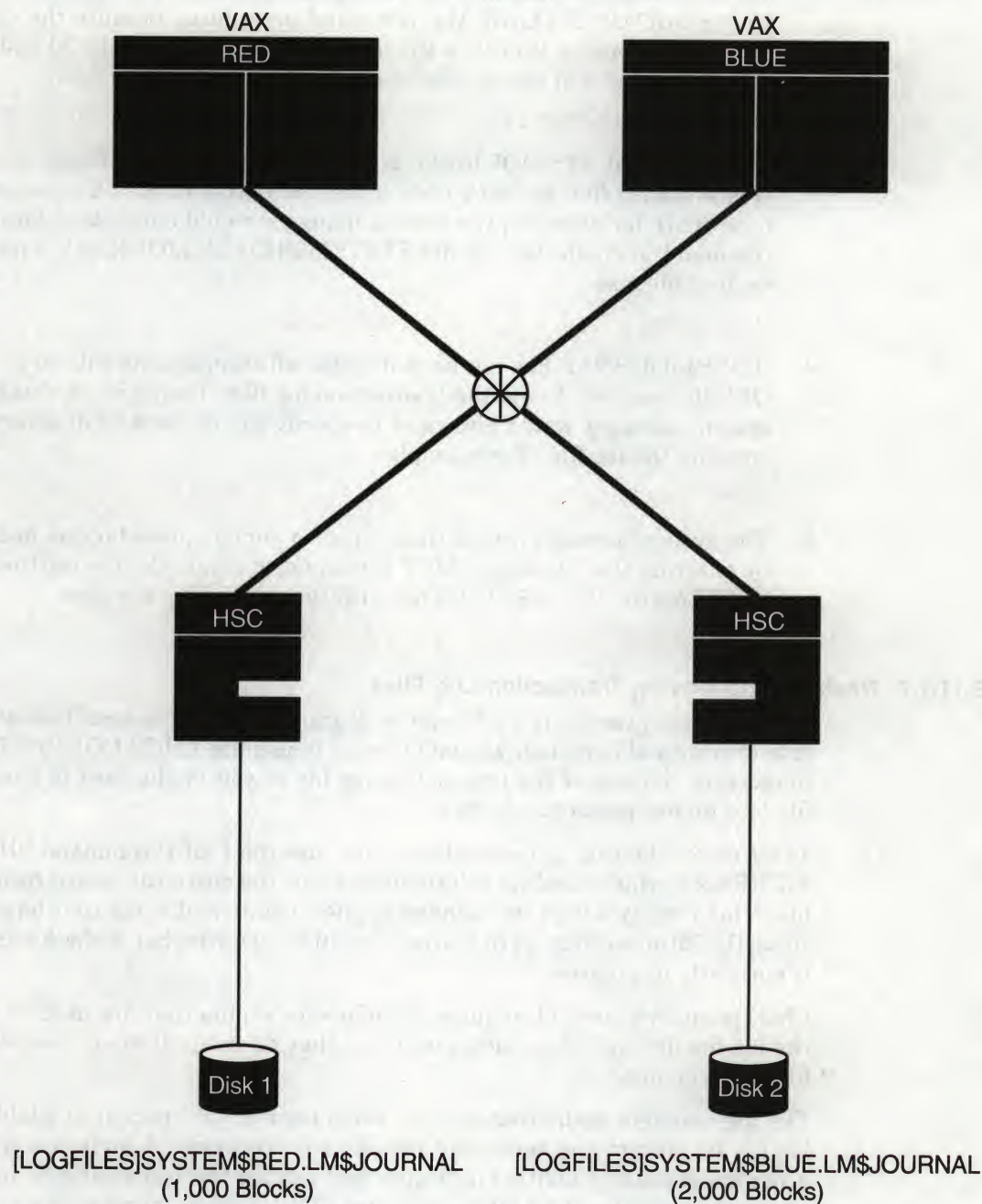
Note

To use LMCP commands, you must have `SYSPRV` privilege. To use the LMCP command `CONVERT`, you must have `CMKRNL` privilege. It is assumed throughout this section that system managers or other individuals who have these privileges will be implementing the procedures described herein.

In this example, the conditions are as follows:

- The sample VAXcluster consists of two nodes, RED and BLUE, with shared access to the devices named DISK1 and DISK2.
- The system manager wants to set up an initial configuration of transaction log files that allows DECdtm services to perform VAXcluster failover.
- The system manager needs to create two log files, one for each node.
- The system manager has determined that the initial log file size will be 1000 blocks on node RED and 2000 blocks on node BLUE. Figure B-9 shows the desired configuration.

Figure B-9 Sample Transaction Log File Configuration on Two-Node VAXcluster



ZK-1894A-GE

Based on the conditions established for this example, the system manager would follow these steps to configure the VAXcluster:

1. On node RED, the system manager would establish a search list for log files by adding the following line to the SYS\$STARTUP:SYLOGICALS command procedure:

```
$ DEFINE/SYSTEM/EXEC SYS$JOURNAL DISK1:[LOGFILES], DISK2:[LOGFILES]
```


Example of Creating a Transaction Log File

2. On node BLUE, the system manager would define a similar search list for transaction log files by adding the following line to the SYS\$STARTUP:SYLOGICALS command procedure. Because the CREATE command creates a log file in the first directory pointed to by SYS\$JOURNAL, this search list will specify the local node log file directory first.

```
$ DEFINE/SYSTEM/EXEC SYS$JOURNAL DISK2:[LOGFILES], DISK1:[LOGFILES]
```

3. Assuming that SYS\$JOURNAL is defined, the system manager would then create the log files for each node using the LMCP CREATE command. On node RED, for example, the system manager would enter the following LMCP command to create the log file SYSTEM\$RED.LM\$JOURNAL with the desired file size:

```
LMCP> CREATE LOGFILE/SIZE=1000 SYSTEM$RED
```

4. If SYS\$JOURNAL has not been defined, all transactions will abort until the DECdtm services locate the transaction log file. Therefore, in this case, the system manager would also need to specify the device and directory when creating the log file. For example:

```
LMCP> CREATE LOGFILE/SIZE=1000 DISK1:[LOGFILES]SYSTEM$RED
```

5. The system manager would then repeat a similar procedure on node BLUE by entering the following LMCP command to create the transaction log file SYSTEM\$BLUE.LM\$JOURNAL with the desired log file size:

```
LMCP> CREATE LOGFILE/SIZE=2000 SYSTEM$BLUE
```

B.11.1.7 Resizing and Moving Transaction Log Files

If transaction processing performance degrades on your system (indicated by the rate of transaction stalls), you might need to use the LMCP CONVERT command to increase the size of the transaction log file or you might need to move the log file to a higher performance disk.

To check for the rate of transaction stalls, use the LMCP command SHOW LOG /CURRENT, which displays information about the currently active transaction log file. This display shows the number of checkpoints and stalls that have occurred since DECdtm services were started and indicates whether a checkpoint or stall is currently in progress.

Checkpoints are normal, regular, log manager events that are used to maintain the log file during transaction execution; they do not indicate degradation in log file performance.

The log manager stalls transactions when insufficient space is available in the log file for correct and successful transaction execution. A high rate of stalls or a permanent stall condition indicates that the log file size should be increased. In such a case, use the LMCP command CONVERT to increase the size of the log file. Occasional stall events might be caused by transitory system activities such as VAXcluster transition events and do not necessarily indicate a permanent shortage of space in the log file.

You can also use the Monitor Utility to check for transaction processing degradation.

The necessary capacity for a log file depends on the number of simultaneous transactions and other factors. Because these factors are variable, Digital cannot recommend the amount of increased size for a transaction log file. You should estimate the percentage of increased transaction workload that caused the log to stall.

Prior to moving or resizing a log file, the system manager must do the following:

1. Disable the transaction log file.

The log file should be disabled before the system is rebooted so that DECdtm services will not reopen the log file after the reboot. The recommended method of disabling a log file is to rename it so that it cannot be found by DECdtm services. Rename the log file with the file type LM\$OLD. For example, if the original log file is called SYS\$JOURNAL:SYSTEM\$ORANGE.LM\$JOURNAL, it should be renamed SYS\$JOURNAL:SYSTEM\$ORANGE.LM\$OLD.

2. Reboot the system.

A reboot is necessary because DECdtm services are an integral part of the VMS executive and cannot be started or stopped independently of the VMS operating system. Because of this requirement, serious considerations should be given to the initial configuration of log files.

After these steps have been completed successfully, the system manager must perform the following conversion procedure to change the size of the transaction log file:

1. Use the LMCP command CONVERT to move the transaction records from the old log file to the new log file and increase its size. Name the new file SYSTEM\$node-name.LM\$JOURNAL.
2. If the conversion is successful, delete the old log file.

The system manager can move the log file to an alternate location by following these steps:

1. Edit SYS\$STARTUP:SYLOGICALS.COM on all nodes in the VAXcluster to include a new definition for the logical name SYS\$JOURNAL, as follows:
\$ DEFINE/SYSTEM/EXEC SYS\$JOURNAL device:[directory]
2. Reboot the system.
3. Copy the log file to the new location, using the following command format:
COPY DEVICE:[DIRECTORY]SYSTEM\$node-name.LM\$OLD-
_SYS\$JOURNAL:SYSTEM\$node-name.LM\$JOURNAL
4. If the copy is successful, delete the old log file.

B.11.2 Format of Transaction Log Files

A transaction log file consists of a file header, section headers, and transaction records.

A log file header contains information about the log file, such as its version number, size, unique identifier, and checkpoints. Checkpoints are mechanisms that bound the search for active transaction records. Therefore, in the event of a system failure, the log manager can efficiently locate the active transaction records needed for system recovery. (An active transaction is one that has not completed.)

A log file is organized into sections and each section has a section header containing information about its own characteristics. This information is used by the log manager to find and read transaction records efficiently.

VMS Version 5.4 Features

Format of Transaction Log Files

The transaction record header identifies the record number and information about the transaction, such as the transaction's state and its unique transaction identifier (TID). A transaction can be in any of three states:

- **PREPARED**—The transaction is in a state where it can be either committed or rolled back.
- **COMMITTED**—The transaction manager has enough information to complete the transaction even though the participants in the transaction have not finished all their operations.
- **FORGOTTEN**—The participants have enough information to complete processing the transaction and will no longer ask about the transaction. Therefore, the transaction can be forgotten.

The transaction record data gives information about the DECdtm version number, the log identifier, and the name and type of resource manager the transaction is involved with.

Example B-2 shows a portion of a sample transaction log file.

Example B-2 Sample Transaction Log File

```
Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM$BLUE.LM$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 9D519DC0-698E-0092-DF95-00000000B20D (21-JUN-1989 09:19:44.54)
Penultimate Checkpoint: 00000012C45E 005E
Last Checkpoint: 000000133E39 0039

Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM$BLUE.LM$JOURNAL;1
Present Length: 166 (000000A6) Last Length: 512 (00000200)
VBN Offset: 2503 (000009C7) Virtual Block: 2505 (000009C9)
Section: 4 (00000004)

Record number 3 (00000003), 77 (004D) bytes
Transaction state (1): PREPARED
Transaction ID: 2B065A40-6E88-0092-EC42-00000000B208 (27-JUN-1989 17:16:11.62)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID:00000000-0000-0000-0000-000000000000
Name (6): "SERVER"
Type (4): PARENT NODE Log ID:6900BC00-6B4F-0092-C8BD-00000000B208
Name (10): "SYSTEM$RED"
```

- ① **Log header**—Contains information about the log's characteristics.
- ② **Section header**—The section header of multiple transaction records.
- ③ **Record number**—A unique record number in decimal and hexadecimal.
- ④ **Record size**—The record size in decimal and hexadecimal.
- ⑤ **Transaction state**—The three states a transaction can be in are PREPARED, COMMITTED, and FORGOTTEN.
- ⑥ **Transaction ID (TID)**—Each transaction has its own unique transaction identifier assigned by the transaction manager.

- ⑦ DECdtm services version number—The software version number of DECdtm services.
- ⑧ Participant type—The types of participant in the transaction.
Participant types include:
 - CHILD NODE—A subordinate transaction manager
 - PARENT NODE—The immediate parent transaction manager
 - LOCAL RM—The recoverable resource manager on the local node
- ⑨ Participant name—The name of the participant in the transaction, also given in hexadecimal.
- ⑩ Log ID—A unique hexadecimal log identifier the participant uses to write its own recovery records.

In Example B-2, the fields labeled ① comprise the log header, ② comprise the section header, ③ through ⑥ comprise the record header, and ⑦ through ⑩ comprise the record data.

LMCP Usage Summary

The Log Manager Control Program is a VMS utility that lets you create and maintain log files of transaction records.

Format

```
$ RUN SYS$SYSTEM:LMCP
```

Usage Summary

To invoke LMCP, enter the following DCL command:

```
$ RUN SYS$SYSTEM:LMCP
```

LMCP returns the following prompt:

```
LMCP>
```

At the LMCP> prompt, you can enter LMCP commands. To exit LMCP, enter EXIT at the LMCP> prompt, or press Ctrl/Z.

You can also execute a single LMCP command by using a DCL string assignment statement, as shown in the following example:

```
$ LMCP := $LMCP  
$ LMCP SHOW LOGFILE SYSTEM$YELLOW
```

In this example, LMCP executes the SHOW command and returns control to DCL.

To use LMCP commands, you must have SYSPRV privilege. To use the LMCP command CONVERT, you must have CMKRNL privilege.

LMCP Commands

This section describes the following LMCP commands and provides examples of how to use them:

- CONVERT
- CREATE
- DUMP
- HELP
- REPAIR, including the following REPAIR subcommands:
 - ABORT
 - COMMIT
 - EXIT
 - FORGET
 - HELP
 - NEXT
- SHOW

Note

To use LMCP commands, you must have SYSPRV privilege. To use the LMCP command CONVERT, you must have CMKRNL privilege. It is assumed throughout this section that system managers or other individuals who have these privileges will be implementing the procedures described herein.

You can abbreviate any command, parameter, or qualifier as long as the abbreviation is unique.

CONVERT

Converts a log file on a given node by transferring the active transaction records from the specified source log file to the specified destination log file. To use the CONVERT command, you need CMKRNL privilege.

Format

```
CONVERT LOGFILE  source_filespec  
                  destination_filespec  
                  [qualifier...]
```

Parameter

source_filespec

Specifies the file specification of the log file from which active transaction records are to be copied.

destination_filespec

Specifies the file specification of the log file where active transaction records are to be written.

Qualifiers

/OWNER=owner_id

Associates an owner or user identification code (UIC) with the log file to be created. You specify the UIC using the standard UIC format as described in the *VMS DCL Concepts Manual*. The default UIC is one of the following:

- The owner UIC of an existing version of the file if the file creator has extended privileges
- The owner UIC of the parent directory if the file creator has extended privileges
- The owner UIC of the creator

/SIZE=file_size

Specifies the size of the log file in blocks. The minimum log file size is 100 blocks.

Description

Use the CONVERT command to resize a log file. For example, if transaction processing performance degrades on your system, then you may need to increase the log file size. See Section B.11.1.7 for more information about resizing and moving log files.

Example

```
LMCP> CONVERT LOGFILE SYSTEM$RED.LM$OLD SYSTEM$RED/SIZE=8000
```

This command transfers all active transaction records from the log file SYSTEM\$RED.LM\$OLD to SYSTEM\$RED and specifies a log file size of 8000 blocks.

CREATE

Creates a log file for a specific node.

Format

CREATE LOGFILE filespec [qualifier...]

Parameter

filespec

Specifies the file specification of the log file to be created. DECdtm services expect the file name to be in the format *SYSTEM\$node-name*, where *node-name* is the name of the node that will use the log file.

Qualifiers

/NEW_VERSION

Creates a new version of a log file if a log file with an identical specification already exists. The new log file is created with the same name and type but with a version number one higher than the highest existing version. Note that, once the new version of the transaction log file is created, then any transaction records in the previous log cannot be accessed.

If the */NEW_VERSION* qualifier is specified for a log file that does not exist, no new file will be created. Instead, an error will be returned.

/OWNER=owner_id

Associates an owner or user identification code (UIC) with the log file to be created. Specify the UIC using the standard UIC format as described in the *VMS DCL Concepts Manual*. The default UIC will be one of the following:

- The owner UIC of an existing version of the file if the file creator has extended privileges
- The owner UIC of the parent directory if the file creator has extended privileges
- The owner UIC of the creator

/SIZE=file_size

Specifies the size of the log file in blocks. The minimum log file size is 100 blocks, and the default log file size is 4000 blocks.

Description

By default, log files are created in the directory specified by *SYS\$JOURNAL*, with a file type of *LM\$JOURNAL* and a size of 4000 blocks. To identify the name of the node that will use the log file, the file name must be in the following format:

SYSTEM\$node-name

Examples

1. LMCP> **CREATE LOGFILE SYSTEM\$BLUE/OWNER=GONZALES/SIZE=4400**

This command creates a log file called **SYSTEM\$BLUE.LM\$JOURNAL**, associates it with user **GONZALES**, and specifies a file size of 4400 blocks.

2. LMCP> **CREATE LOGFILE SYSTEM\$YELLOW/OWNER=[USER,FRED]/SIZE=4000**

This command creates a log file called **SYSTEM\$YELLOW.LM\$JOURNAL**, associates it with the UIC group **USER**, member **FRED**, and specifies a log file size of 4000 blocks.

3. LMCP> **CREATE LOGFILE SYSTEM\$BLUE/NEW_VERSION/OWNER=GONZALES/SIZE=4400**

This command creates a new log file that supersedes the current highest version of **SYSTEM\$BLUE.LM\$JOURNAL** and is given a version number one higher. Also, the new log file is associated with user **GONZALES** and specifies a file size of 4400 blocks.

DUMP

Displays (or “dumps”) the contents of a specified log file.

Format

DUMP filespec [qualifier...]

Parameter

filespec

Specifies the file specification of the log file.

Qualifiers

/ACTIVE

Specifies that only records relating to active transactions within the log file are to be displayed.

/FORMAT(default)

/NOFORMAT

Displays the contents of the log file as formatted records. If the /NOFORMAT qualifier is specified, only the log file header is displayed.

/HEX

Specifies that the contents of the log file dump are displayed as ASCII characters and hexadecimal longwords. Use both the /NOFORMAT and /HEX qualifiers to format a DUMP operation in hexadecimal only.

/LOGID=log_identifier

Specifies the log identifier, in hexadecimal format, associated with a specific resource manager. The /LOGID qualifier can be used only in conjunction with the /RM qualifier.

/OUTPUT[=filespec]

Specifies that the output is written to the file specified. By default, the DUMP command writes the output to SYS\$OUTPUT. If you enter /OUTPUT with no file specification, LMCP_DUMP is the default file name and LIS is the default type.

/RM=rm_identifier

Selects the transactions to be displayed according to the resource manager participating in the transaction. The argument supplied for the **rm_identifier** can be either the ASCII character string for the resource manager name or its hexadecimal equivalent. When specifying a hexadecimal string, you must prefix the characters %X to the hexadecimal string.

If a partial resource manager name is supplied as the argument for the **rm_identifier**, LMCP selects all resource managers having names that begin with the supplied string.

/STATE=transaction_state

Selects the transactions to be displayed according to their transaction states. A value of either PREPARED or COMMITTED can be supplied as an argument to the /STATE qualifier. If the /STATE qualifier is not supplied, all transactions records are selected.

/TID=transaction_id

Selects the transactions to be displayed according to the transaction identifier. The argument supplied for the **transaction_id** must be a hexadecimal character string.

Description

If you entered the DUMP command, the contents of the log file you specified are displayed. By default the log file records are displayed as formatted records.

Examples

1. LMCP> DUMP SYSTEM\$BLUE/HEX/NOFORMAT

```
Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM$BLUE.LM$JOURNAL;2
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 9D519DC0-698E-0092-DF95-00000000B20D (21-JUN-1989 09:19:44.54)
Penultimate Checkpoint: 00000012C45E 005E
Last Checkpoint: 000000133E39 0039
```

```
Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM$BLUE.LM$JOURNAL;2
Present Length: 68 (00000044) Last Length: 512 (00000200)
VBN Offset: 2504 (000009C8) Virtual Block: 2506 (000009CA)
Section: 3 (00000003)

Record number 1 (00000001), 48 (0030) bytes
Transaction state (2): COMMITTED
Transaction ID: 2B065A40-6E88-0092-EC42-00000000B208 (27-JUN-1989 17:16:11.62)
01000000 00B20842 EC00926E 882B065A 40020030 0..@Z+.n..iB.²..... 0000
00060000 00000000 00000000 00000000 00000300 ..... 0014
00305245 56524553 SERVER0. 0028
```

```
Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM$BLUE.LM$JOURNAL;2
Present Length: 166 (000000A6) Last Length: 512 (00000200)
VBN Offset: 2503 (000009C7) Virtual Block: 2505 (000009C9)
Section: 4 (00000004)

Record number 3 (00000003), 77 (004D) bytes
Transaction state (1): PREPARED
Transaction ID: 2B065A40-6E88-0092-EC42-00000000B208 (27-JUN-1989 17:16:11.62)
01000000 00B20842 EC00926E 882B065A 4001004D M..@Z+.n..iB.²..... 0000
00060000 00000000 00000000 00000000 00000300 ..... 0014
00B208BD C800926B 4F6900BC 00045245 56524553 SERVER..¼.iok..È½.². 0028
00 4D444552 244D4554 53595300 0A000000 .....SYSTEM$REDM. 003C
```

```
Record number 2 (00000002), 21 (0015) bytes
Transaction state (0): FORGOTTEN
Transaction ID: 2A6DC3C0-6E88-0092-EC42-00000000B208 (27-JUN-1989 17:16:10.62)
15000000 00B20842 EC00926E 882A6DC3 C0000015 ...Ãm*.n..iB.²..... 0000
00 . 0014
```

```
Record number 1 (00000001), 48 (0030) bytes
Transaction state (2): COMMITTED
Transaction ID: 2A6DC3C0-6E88-0092-EC42-00000000B208 (27-JUN-1989 17:16:10.62)
01000000 00B20842 EC00926E 882A6DC3 C0020030 0..Ãm*.n..iB.²..... 0000
00060000 00000000 00000000 00000000 00000300 ..... 0014
00305245 56524553 SERVER0. 0028
```

·
·
·

This command produces a dump—in hexadecimal format—of the specified log file.

LMCP DUMP

2. LMCP> DUMP SYSTEM\$BLUE/HEX/OUTPUT=EXAMPLE

This command writes a dump—in hexadecimal format—of the specified log file to the file **EXAMPLE.LIS**.

3. LMCP> DUMP SYSTEM\$PURPLE/ACTIVE

Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM\$PURPLE.LM\$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 2F99A820-BAB2-0092-9310-00000000B1FE (2-OCT-1989 15:28:26.53)
Penultimate Checkpoint: 000000000000 0000
Last Checkpoint: 000000010BD9 01D9

Transaction state (2): COMMITTED
Transaction ID: 84C67760-BAB2-0092-8243-00000000B1FE (2-OCT-1989 15:30:49.43)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (11): "THREAD_5.29" (39322E 355F4441 45524854)
Type (2): CHILD NODE Log ID: 748FF0C0-B52A-0092-9011-00000000B204
Name (13): "SYSTEM\$ORANGE" (45 474E4152 4F244D45 54535953)

Transaction state (2): COMMITTED
Transaction ID: 84C1E380-BAB2-0092-8243-00000000B1FE (2-OCT-1989 15:30:49.40)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (11): "THREAD_4.29" (39322E 345F4441 45524854)
Type (2): CHILD NODE Log ID: 748FF0C0-B52A-0092-9011-00000000B204
Name (13): "SYSTEM\$ORANGE" (45 474E4152 4F244D45 54535953)

Total of 2 transactions active, 0 prepared and 2 committed.

This command displays a dump of all active transactions of the specified log file.

4. LMCP> DUMP SYSTEM\$GREEN/STATE=PREPARED

Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM\$GREEN.LM\$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 748FF0C0-B52A-0092-9011-00000000B204 (25-SEP-1989 14:34:14.86)
Penultimate Checkpoint: 00000002DDB7 01B7
Last Checkpoint: 00000002FC41 0241

Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM\$GREEN.LM\$JOURNAL;1
Present Length: 169 (000000A9) Last Length: 512 (00000200)
VBN Offset: 380 (0000017C) Virtual Block: 382 (0000017E)
Section: 2 (00000002)

Record number 3 (00000003), 80 (0050) bytes
Transaction state (1): PREPARED
Transaction ID: F30CAF60-BA84-0092-8FA6-00000000B24B (2-OCT-1989 10:04:37.59)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (6): "SERVER" (5245 56524553)
Type (4): PARENT NODE Log ID: 68165820-BA84-0092-FC95-00000000B24B
Name (13): "SYSTEM\$ORANGE" (45 474E4152 4F244D45 54535953)

Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM\$GREEN.LM\$JOURNAL;1
Present Length: 100 (00000064) Last Length: 512 (00000200)
VBN Offset: 379 (0000017B) Virtual Block: 381 (0000017D)
Section: 3 (00000003)

Record number 1 (00000001), 80 (0050) bytes
Transaction state (1): PREPARED
Transaction ID: F2F8D940-BA84-0092-8FA6-00000000B24B (2-OCT-1989 10:04:37.46)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (6): "SERVER" (5245 56524553)
Type (4): PARENT NODE Log ID: 68165820-BA84-0092-FC95-00000000B24B
Name (13): "SYSTEM\$ORANGE" (45 474E4152 4F244D45 54535953)
.
.
.
Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM\$GREEN.LM\$JOURNAL;1
Present Length: 100 (00000064) Last Length: 0 (00000000)
VBN Offset: 0 (00000000) Virtual Block: 2 (00000002)
Section: 376 (00000178)

Record number 1 (00000001), 80 (0050) bytes
Transaction state (1): PREPARED
Transaction ID: 809D5600-BA84-0092-8FA6-00000000B24B (2-OCT-1989 10:01:25.60)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (6): "SERVER" (5245 56524553)
Type (4): PARENT NODE Log ID: 68165820-BA84-0092-FC95-00000000B24B
Name (13): "SYSTEM\$ORANGE" (45 474E4152 4F244D45 54535953)

This command displays a dump of all prepared records of the specified log file.

5. LMCP> DUMP SYSTEM\$GREEN/TID=FAC21DE2-BA88-0092-8FA6-00000000B24B/ACTIVE

Dump of log file DISK1:[MASTER.JOURNALS]SYSTEM\$GREEN.LM\$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 68165820-BA84-0092-FC95-00000000B24B (2-OCT-1989 10:00:44.45)
Penultimate Checkpoint: 0000000711D3 13D3
Last Checkpoint: 000000072742 1542

Transaction state (2): COMMITTED
Transaction ID: FAC21DE2-BA88-0092-8FA6-00000000B24B (2-OCT-1989 10:33:28.51)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (11): "THREAD_13.4" (342E33 315F4441 45524854)

Total of 2 transactions active, 0 prepared and 2 committed.

**This command displays a dump of the record for the specified active transaction.
(If the transaction is not active, only the active transaction count number is
displayed.)**

HELP

Provides information about LMCP commands and parameters.

Format

HELP [help-topic [help-subtopic]]

Parameter

help-topic

Specifies the command to be explained.

help-subtopic

Specifies the qualifier to be explained.

Examples

1. LMCP> **HELP**

Information available:

CONVERT	CREATE	Description	DUMP	EXIT	HELP
REPAIR	SHOW				

This command invokes help and displays all commands for which further information exists.

2. LMCP> **HELP CREATE**

CREATE

Creates a log file.

Format:

CREATE LOGFILE filespec [qualifier...]

Additional information available:

filespec qualifiers

/OWNER /SIZE

Example

This command provides a description of the CREATE command.

REPAIR

Selects records within a log file so that transactions can be repaired by having their transaction states changed. Once the transaction records have been selected, REPAIR subcommands can be used to change the transaction states.

Note

Because the REPAIR command lets you change transaction states locally without regard to the global state, you must use this command with caution. If you do not change all necessary characteristics of a transaction record, the transaction could be placed in an inconsistent state, resulting in potential data loss.

Format

REPAIR filespec [qualifier...]

Parameter

filespec

Specifies the file specification of the log file containing the transaction records to be repaired.

Qualifiers

/LOGID=log_identifier

Specifies the log identifier, in hexadecimal format, associated with a specific resource manager. The /LOGID qualifier can be used only in conjunction with the /RM qualifier.

/RM=rm_identifier

Selects the transactions to be repaired according to the resource manager participating in the transaction. The argument supplied for the **rm_identifier** can be either the ASCII character string for the resource manager name or its hexadecimal equivalent. When specifying a hexadecimal string, you must prefix the characters %X to the hexadecimal string.

If a partial resource manager name is supplied as the argument for the **rm_identifier**, LMCP selects all resource managers having names that begin with the supplied string.

/STATE=transaction_state

Selects the transactions to be repaired according to their transaction states. A value of either PREPARED or COMMITTED can be supplied as an argument to the /STATE qualifier. If the /STATE qualifier is not supplied, all active transactions (both PREPARED and COMMITTED) are selected.

/TID=transaction_id

Selects the transactions to be repaired according to the transaction identifier. The argument supplied for the **transaction_id** must be a hexadecimal character string.

Description

The REPAIR command allows you to manually modify active transaction records in a log file.

When you enter the REPAIR command, LMCP enters the REPAIR command mode and produces a listing of the log file's contents, as selected by the specified REPAIR command qualifier. Each transaction record is displayed sequentially so that you can modify its characteristics. After each record in the filtered log file is displayed, the REPAIR> prompt returns. You can then enter REPAIR subcommands to change the transaction states of specific records. The REPAIR subcommands are as follows:

ABORT
COMMIT
EXIT
FORGET
HELP
NEXT

Once you finish modifying a transaction record, you can use the REPAIR subcommand NEXT to advance to the next sequential record in the file.

To return to the LMCP> prompt, you must exit the REPAIR command mode by entering the EXIT subcommand or by pressing Ctrl/Z.

The sections that follow the REPAIR command examples describe each of the REPAIR subcommands.

Examples

1. LMCP> REPAIR SYSTEM\$ORANGE/STATE=PREPARED/RM=LOGL

This command selects all PREPARED transaction records in the log file SYSTEM\$ORANGE. It specifies that only records from participating resource managers having names beginning with LOGL are to be selected.

2. LMCP> REPAIR SYSTEM\$ORANGE/RM=LOGLOAD -
_LMCP> /LOGID=68165820-BA84-0092-FC95-00000000B24B

This command selects all active transaction records in the log file SYSTEM\$ORANGE. It specifies that only records with a participating resource manager called LOGLOAD and associated log identifier of 68165820-BA84-0092-FC95-00000000B24B are to be selected.

3. LMCP> REPAIR SYSTEM\$ORANGE/RM=%X534552564552

This command selects all active transaction records in the log file SYSTEM\$ORANGE. It specifies that only records from a participating resource manager with a hexadecimal name 534552564552 are to be selected.

4. LMCP> REPAIR SYSTEM\$ORANGE -
_LMCP> /TID=8C689380-BA84-0092-8FA6-00000000B24B

This command selects the active transaction record in the log file SYSTEM\$ORANGE. It specifies that only the record for the transaction with a hexadecimal TID 8C689380-BA84-0092-8FA6-00000000B24B is to be selected.

ABORT

Changes the state of a transaction from PREPARED to ABORTED.

Format

ABORT

Example

```
LMCP> REPAIR SYSTEM$RED
```

```
Dump of log file DISK$MASTER:[MASTER.JOURNALS]SYSTEM$RED.LM$JOURNAL;1
```

```
End of file block 4000 / Allocated 4000
```

```
Log Version 1.0
```

```
Log File UID: 748FF0C0-B52A-0092-9011-00000000B204 (25-SEP-1989 14:34:14.86)
```

```
Penultimate Checkpoint: 000000073E2D 042D
```

```
Last Checkpoint: 000000077D7C 037C
```

```
Transaction state (1): PREPARED
```

```
Transaction ID: FACFD981-BA88-0092-8FA6-00000000B24B ( 2-OCT-1989 10:33:28.60)
```

```
DECdtm services V1.0
```

```
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
```

```
Name (6): "SERVER" (5245 56524553)
```

```
Type (4): PARENT NODE Log ID: 68165820-BA84-0092-FC95-00000000B24B
```

```
Name (13): "SYSTEM$ORANGE" (45 474E4152 4F244D45 54535953)
```

```
REPAIR> ABORT
```

```
REPAIR> EXIT
```

```
LMCP>
```

The initial REPAIR command selects all active transaction records in the log file SYSTEM\$RED. The ABORT subcommand changes the state of the presented transaction from PREPARED to ABORTED. The EXIT subcommand exits the REPAIR command mode.

REPAIR COMMIT

COMMIT

Changes the state of a transaction from PREPARED to COMMITTED.

Format

COMMIT

Example

```
LMCP> REPAIR SYSTEM$RED
```

```
Dump of log file DISK$MASTER:[MASTER.JOURNALS]SYSTEM$RED.LM$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 748FF0C0-B52A-0092-9011-00000000B204 (25-SEP-1989 14:34:14.86)
Penultimate Checkpoint: 000000073E2D 042D
Last Checkpoint: 000000077D7C 037C
```

```
Transaction state (1): PREPARED
Transaction ID: FACFD981-BA88-0092-8FA6-00000000B24B ( 2-OCT-1989 10:33:28.60)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (6): "SERVER" (5245 56524553)
Type (4): PARENT NODE Log ID: 68165820-BA84-0092-FC95-00000000B24B
Name (13): "SYSTEM$ORANGE" (45 474E4152 4F244D45 54535953)
REPAIR> COMMIT
REPAIR> EXIT
LMCP>
```

The initial REPAIR command selects all active transaction records in the log file SYSTEM\$RED. The COMMIT subcommand changes the state of the transaction from PREPARED to COMMITTED. The EXIT subcommand exits the REPAIR command mode.

EXIT

Exits the REPAIR command mode and returns the LMCP> prompt.

Format

EXIT

REPAIR FORGET

FORGET

Specifies that a transaction with a state of COMMITTED can be forgotten, which means the committed transaction record can be removed from the log file.

Format

FORGET

Example

```
LMCP> REPAIR SYSTEM$RED
```

```
Dump of log file DISK$MASTER:[MASTER.JOURNALS]SYSTEM$RED.LM$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 748FF0C0-B52A-0092-9011-00000000B204 (25-SEP-1989 14:34:14.86)
Penultimate Checkpoint: 000000073E2D 042D
Last Checkpoint: 000000077D7C 037C
```

```
Transaction state (2): COMMITTED
Transaction ID: F2F8D940-BA84-0092-8FA6-00000000B24B ( 2-OCT-1989 10:04:37.46)
DECdtm Services Log Format V1.0
Type (3): LOCAL RM Log ID: 00000000-0000-0000-0000-000000000000
Name (10): "THREAD_6.4" (342E 365F4441 45524854)
REPAIR> FORGET
REPAIR> NEXT
```

The initial REPAIR command selects all active transaction records in the log file SYSTEM\$RED. The FORGET subcommand specifies that the transaction can be forgotten. The NEXT subcommand advances to the next record.

HELP

Provides information about REPAIR subcommands and parameters.

Format

HELP [help-topic [help-subtopic]]

Parameter

help-topic

Specifies the subcommand to be explained.

help-subtopic

Specifies the qualifier to be explained.

Examples

1. REPAIR> **HELP**

REPAIR

SUBCOMMANDS

Entering the REPAIR command produces a listing of the log file's contents, as selected by the optional REPAIR command qualifiers. Each transaction record is displayed sequentially, so that a user can modify its characteristics.

After each record in the filtered log file is displayed, the REPAIR> prompt is returned. A user can then issue REPAIR subcommands to change the transaction states of specific records. A user must issue a NEXT subcommand to advance to the next sequential record in the file.

To return to the LMCP> prompt, a user must exit the REPAIR command mode by entering the EXIT subcommand or by pressing Ctrl/Z.

Additional information available:

ABORT COMMIT EXIT FORGET NEXT

This command invokes help and displays all subcommands for which further information exists.

2. REPAIR> **HELP ABORT**

REPAIR

SUBCOMMANDS

ABORT

Changes the state of a transaction from PREPARED to ABORTED.

Format:

ABORT

This command provides a description of the ABORT subcommand.

NEXT

Advances to the next record in a transaction log.

Format

NEXT

SHOW

Lists information about transaction log files.

Format

SHOW LOGFILE filespec [qualifier...]

Parameter

filespec

Specifies one or more log files to be listed. The syntax of the file specification determines which files will be listed, as follows:

- If you enter a file name or a file name containing a wildcard character, the SHOW command lists each file matching the name specified.
- If you do not enter a file specification, the SHOW command lists all log files in the directory SYS\$JOURNAL.

QUALIFIER

/CURRENT

Specifies that information about the currently active log file is shown. This information includes the number of checkpoints and stalls that have occurred since DECdtm services were started up and indicates whether a checkpoint or stall is currently in progress.

Note that no file specification is necessary when the /CURRENT qualifier is used.

/FULL

Lists all log file attributes.

/OUTPUT[=filespec]

Specifies that the output be written to the file specified. By default, the SHOW command writes the output to SYS\$OUTPUT. If you enter /OUTPUT with no file specification, then LMCP_SHOW is the default file name and LIS is the default type.

Description

The SHOW command produces a list of existing log files matching the selection criteria specified. The asterisk and percent sign wildcard characters can be passed to the SHOW command to represent file names.

Examples

1. LMCP> SHOW LOGFILE SYSTEM\$B*/FULL

Directory of DISK\$MASTER: [MASTER.JOURNALS]

DISK\$MASTER: [MASTER.JOURNALS]SYSTEM\$BLUE.LM\$JOURNAL;1

End of file block 4000 / Allocated 4000

Log Version 1.0

Log File UID: 275300C0-7A71-0092-D3A8-00000000B232 (12-JUL-1989 21:01:40.94)

Penultimate Checkpoint: 0000CE644AF2 02F2

Last Checkpoint: 0000CE6457F2 03F2

VMS Version 5.4-3 Features

SHOW

DISK\$MASTER:[MASTER.JOURNALS]SYSTEM\$BLACK.LM\$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 9D519DC0-698E-0092-DF95-00000000B20D (21-JUN-1989 09:19:44.54)
Penultimate Checkpoint: 00000012C45E 005E
Last Checkpoint: 000000133E39 0039

DISK\$MASTER:[MASTER.JOURNALS]SYSTEM\$BRONZE.LM\$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: 21847980-5F78-0092-3F5D-00000000B1FF (8-JUN-1989 13:13:36.28)
Penultimate Checkpoint: 000000ECADE5 41E5
Last Checkpoint: 000000F105FC 41FC

DISK\$MASTER:[MASTER.JOURNALS]SYSTEM\$BROWN.LM\$JOURNAL;1
End of file block 4000 / Allocated 4000
Log Version 1.0
Log File UID: A6173DC0-3DE2-0092-0000-00000000B1FF (26-APR-1989 19:30:25.82)
Penultimate Checkpoint: 00000C8B4819 2019
Last Checkpoint: 00000C8BC15B 335B

Total of 4 files.

This command lists all log files with file names beginning with SYSTEM\$B.

2. LMCP> SHOW LOGFILE
Directory of DISK1:[MASTER.LOGFILES]

SYSTEM\$BLACK.LM\$JOURNAL;1
SYSTEM\$BLUE.LM\$JOURNAL;1

Total of 2 files.

Directory of DISK1:[MASTER.NAMES]

SYSTEM\$GREEN.LM\$JOURNAL;1
SYSTEM\$ORANGE.LM\$JOURNAL;1
SYSTEM\$RED.LM\$JOURNAL;1

Total of 3 files.

Grand total of 2 directories 5 files.

This command lists all directories equivalent to SYS\$JOURNAL and their log files.

3. LMCP> SHOW LOGFILE SYSTEM\$RED/FULL/OUTPUT=EXAMPLE

This command lists all percentage information for the specified log file and writes it to the file EXAMPLE.LIS.

4. LMCP> SHOW LOGFILE/CURRENT

Checkpoints started/ended 124/123
Stalls started/ended 1/1
Log status: checkpoint in progress, no stall in progress

This command shows status information about the currently active log file.

B.12 Monitor Utility (MONITOR)

The VMS Monitor Utility (MONITOR) is a system management tool that you can use to obtain information about operating system performance. This section describes the following enhancements to Version 5.4 of the VMS Monitor Utility:

- New MONITOR TRANSACTION command and TRANSACTION class (for use within a DECdtm services environment)
- New MONITOR VECTOR command and VECTOR class (for use within a vector processing environment)

See the *VMS Monitor Utility Manual* for information about other classes and commands.

B.12.1 MONITOR TRANSACTION Command

The MONITOR TRANSACTION command initiates monitoring of the TRANSACTION class, which shows information about transactions coordinated by the DECdtm services. (For a complete description of DECdtm services, see Section B.3.)

Use this command as follows:

1. Invoke the Monitor Utility by entering the DCL command MONITOR. The utility then displays the following prompt:
MONITOR>
2. At the MONITOR> prompt, enter the MONITOR TRANSACTION command. The format, description, and examples of how to use this command follow.

MONITOR TRANSACTION

Format

MONITOR TRANSACTION

Qualifiers

/qualifier[,...]

One or more qualifiers, described as follows:

Class-name qualifiers

/ALL

Specifies that a table of all available statistics (current, average, minimum, and maximum) is to be included in the display and summary output. For summary output, this qualifier is the default for all classes; otherwise, it is the default for all classes except CLUSTER, MODES, PROCESSES, STATES, SYSTEM, and VECTOR.

/AVERAGE

Selects average statistics to be displayed in a bar graph for display and summary output.

/CURRENT

Selects current statistics to be displayed in a bar graph for display and summary output. The /CURRENT qualifier is the default for the CLUSTER, MODES, STATES, SYSTEM, and VECTOR classes.

/MAXIMUM

Selects maximum statistics to be displayed in a bar graph for display and summary output.

/MINIMUM

Selects minimum statistics to be displayed in a bar graph for display and summary output.

Description

The TRANSACTION class consists of the following data items:

- **Start Rate**—The rate at which transactions are started.
- **Prepare Rate**—The rate at which transactions are placed in the prepare state by DECdtm services.
- **One-Phase Commit Rate**—The rate that one-phase commit transactions complete using the one-phase commit operation. This operation, which consumes significantly fewer system resources, is used when there is only a single resource manager participating in the transaction.
- **Total Commit Rate**—The rate at which transactions are committed. This value is the combined total of one-phase and two-phase commit transactions.
- **Abort Rate**—The rate at which transactions are aborted.
- **End Rate**—The rate at which transactions are ended.

- Remote Start Rate—The rate at which transactions are started by a transaction manager on a remote node.
- Remote Add Rate—The rate of remote add branch operations.
- Completion Rate—The rate of completed transactions, indexed by their duration time in seconds. Following is a list of the completion rate categories:

Completion Rate 0-1	The number of transactions completed in 0-1 second (1 second or less)
Completion Rate 1-2	The number of transactions completed in 1-2 seconds
Completion Rate 2-3	The number of transactions completed in 2-3 seconds
Completion Rate 3-4	The number of transactions completed in 3-4 seconds
Completion Rate 4-5	The number of transactions completed in 4-5 seconds
Completion Rate 5+	The number of transactions that took more than 5 seconds to complete

A transaction completed in 0.5 second is included in the count displayed for the Completion Rate 0-1 category, which indicates the number of transactions completed in the last time interval that took 0-1 second to execute. See the example displays that follow.

Examples

1. MONITOR> MONITOR TRANSACTION/ALL

VAX/VMS Monitor Utility					
DISTRIBUTED TRANSACTION STATISTICS					
on node SAMPLE					
16-JAN-1990 14:52:34					
		CUR	AVE	MIN	MAX
Start Rate		34.76	34.76	34.76	34.76
Prepare Rate		33.77	33.77	33.77	33.77
One Phase Commit Rate		0.00	0.00	0.00	0.00
Total Commit Rate		35.09	35.09	35.09	35.09
Abort Rate		0.00	0.00	0.00	0.00
End Rate		35.09	35.09	35.09	35.09
Remote Start Rate		31.12	31.12	31.12	31.12
Remote Add Rate		31.45	31.45	31.45	31.45
Completion Rate	0-1	35.09	35.09	35.09	35.09
by Duration	1-2	0.00	0.00	0.00	0.00
in Seconds	2-3	0.00	0.00	0.00	0.00
	3-4	0.00	0.00	0.00	0.00
	4-5	0.00	0.00	0.00	0.00
	5+	0.00	0.00	0.00	0.00

This example shows the status of all transactions on node SAMPLE.

VMS Version 5.4 Features MONITOR TRANSACTION

2. MONITOR> MONITOR TRANSACTION/MAXIMUM

```

                                VAX/VMS Monitor Utility
                                DISTRIBUTED TRANSACTION STATISTICS
                                on node SAMPLE
                                16-JAN-1990 14:51:04
                                0      25      50      75      100
                                + - - - - + - - - - + - - - - +
Start Rate                    35 | *****
Prepare Rate                  37 | *****
One Phase Commit Rate        35 | *****
Total Commit Rate             35 | *****
Abort Rate                    35 | *****
End Rate                      33 | *****
Remote Start Rate             32 | *****
Remote Add Rate               32 | *****
Completion Rate 0-1           35 | *****
by Duration 1-2               |
in Seconds 2-3                |
3-4                           |
4-5                           |
5+                             |
                                + - - - - + - - - - + - - - - +

```

This example shows the maximum statistics of all transactions on node SAMPLE.

B.12.2 TRANSACTION Class Record

The TRANSACTION class record contains data describing the operations of the DECdtm transaction manager. The TRANSACTION class has a record type of 22 and a size of 69 bytes. Figure B-10 illustrates the format of a TRANSACTION class record; Table B-7 describes the contents of each of its fields.

Figure B-10 TRANSACTION Class Record Format

Class Header (14 Bytes)	
Starts	MNR_TRA\$L_STARTS
Prepares	MNR_TRA\$L_PREPARES
One Phase Commits	MNR_TRA\$L_ONE_PHASE
Commits	MNR_TRA\$L_COMMITS
Aborts	MNR_TRA\$L_ABORTS
Ends	MNR_TRA\$L_ENDS
Branches	MNR_TRA\$L_BRANCHS
Adds	MNR_TRA\$L_ADDS
0-1 Transactions	MNR_TRA\$L_BUCKETS1
1-2 Transactions	MNR_TRA\$L_BUCKETS2
2-3 Transactions	MNR_TRA\$L_BUCKETS3
3-4 Transactions	MNR_TRA\$L_BUCKETS4
4-5 Transactions	MNR_TRA\$L_BUCKETS5
5+ Transactions	MNR_TRA\$L_BUCKETS6

ZK-2023A-GE

VMS Version 5.4 Features

TRANSACTION Class Record

Table B-7 Descriptions of TRANSACTION Class Record Fields

Field	Symbolic Offset	Contents
Starts	MNR_TRA\$L_STARTS	Count of transaction operations started. The number of times the system service \$START_TRANS has been successfully completed (longword, C).
Prepares	MNR_TRA\$L_PREPARES	Count of transactions that have been prepared (longword, C).
One Phase Commits	MNR_TRA\$L_ONE_PHASE	Count of one-phase commit events initiated (longword, C).
Commits	MNR_TRA\$L_COMMITS	Count of transactions committed. This is the combined total of one-phase and two-phase commits (longword, C).
Aborts	MNR_TRA\$L_ABORTS	Count of transactions aborted. Combined total of planned and unplanned aborts (longword, C).
Ends	MNR_TRA\$L_ENDS	Count of transactions ended. The number of times \$END_TRANS has successfully completed (longword, C).
Branches	MNR_TRA\$L_BRANCHS	Count of start remote (to a remote parent) branch operations (longword, C).
Adds	MNR_TRA\$L_ADDS	Count of add remote (to a remote subordinate parent) branch operations (longword, C).
0-1 Transactions	MNR_TRA\$L_BUCKETS1	Count of transactions with a duration of less than 1 second (longword, C).
1-2 Transactions	MNR_TRA\$L_BUCKETS2	Count of transactions with a duration of 1 to 2 (1.99) seconds (longword, C).
2-3 Transactions	MNR_TRA\$L_BUCKETS3	Count of transactions with a duration of 2 to 3 seconds (longword, C).
3-4 Transactions	MNR_TRA\$L_BUCKETS4	Count of transactions with a duration of 3 to 4 seconds (longword, C).
4-5 Transactions	MNR_TRA\$L_BUCKETS5	Count of transactions with a duration of 4 to 5 seconds (longword, C).
5+ Transactions	MNR_TRA\$L_BUCKETS6	Count of transactions with a duration greater than 5 seconds (longword, C).

B.12.3 MONITOR VECTOR Command

The MONITOR VECTOR command displays the number of 10-millisecond clock ticks per second in which one or more vector consumers have been scheduled on each currently configured vector processor in the system. Because the VMS operating system schedules vector consumers only on those processors identified as "vector present," the VECTOR class output never displays vector CPU time for those processors that are "vector absent."

Note that, because vector consumers can use either or both the vector CPU and scalar CPU components of a vector-present processor, the vector CPU time in the VECTOR class display is not a strict measure of the actual usage of the processor's vector CPU component. Rather, it indicates the time during which a scheduled vector consumer has reserved both vector CPU and scalar CPU

components of the vector-present processor for its own exclusive use. (For a more complete description of the vector processing environment, see Section B.2.)

Use this command as follows:

1. Invoke the Monitor Utility by entering the DCL command MONITOR. The utility then displays the following prompt:

MONITOR>

2. At the MONITOR> prompt, enter the MONITOR VECTOR command. The format, description, and an example of this command follow.

MONITOR VECTOR

Format

MONITOR VECTOR

Qualifiers

/qualifier[,...]

One or more qualifiers, described as follows:

Class-name qualifiers

/ALL

Specifies that a table of all available statistics (current, average, minimum, and maximum) is to be included in the display and summary output. For summary output, this qualifier is the default for all classes; otherwise, it is the default for all classes except CLUSTER, MODES, PROCESSES, STATES, SYSTEM, and VECTOR.

/AVERAGE

Selects average statistics to be displayed in a bar graph for display and summary output.

/CURRENT

Selects current statistics to be displayed in a bar graph for display and summary output. The /CURRENT qualifier is the default for the CLUSTER, MODES, STATES, SYSTEM, and VECTOR classes.

/MAXIMUM

Selects maximum statistics to be displayed in a bar graph for display and summary output.

/MINIMUM

Selects minimum statistics to be displayed in a bar graph for display and summary output.

Description

The VECTOR class consists of the data item Vector Scheduled Rate, which is represented by a display of statistics that show the rates of 10-millisecond clock ticks per second during which vector consumers have been scheduled on each vector-present CPU.

Example

MONITOR> MONITOR VECTOR

```
VAX/VMS Monitor Utility
VECTOR PROCESSOR STATISTICS
  on node SAMPLE
12-JUN-1991 22:52:42
```

[illegible]

This example shows the VECTOR class display for a multiprocessing system containing two vector-present processors, CPU 0 and CPU 4. Displayed statistics represent rates of 10-millisecond clock ticks per second. For an average of 13 ticks per second over the last collection interval, vector consumers have been scheduled on CPU 0. For an average of 58 ticks per second over the last collection interval, vector consumers have been scheduled on CPU 4.

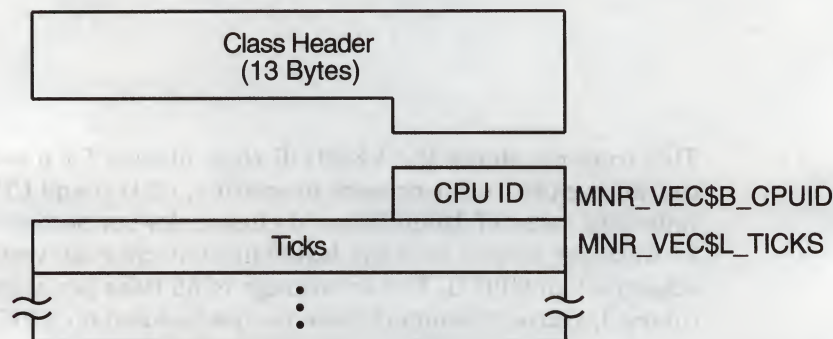
B.12.4 VECTOR Class Record

The VECTOR class record contains data describing the time during which vector consumers have been scheduled on a vector-present processor. Its record type number is 23. A VECTOR class record is of variable length and depends on the number of active processors in the system. Assuming all processors are active, MONITOR calculates its size by adding the size of the class header and the data block, as follows:

$$13 + (5 * \text{MNR_SYI\$B_VPCPUS})$$

Figure B-11 illustrates the format of a VECTOR class record; Table B-8 describes the contents of each of its fields.

Figure B-11 VECTOR Class Record Format



ZK-1942A-GE

Table B-8 Descriptions of VECTOR Class Record Fields

Field	Symbolic Offset	Contents
CPU ID	MNR_VEC\$B_CPUID	Identification of the processor from which the data has been collected (1 byte).
Ticks	MNR_VEC\$L_TICKS	Number of 10-millisecond clock ticks in which a vector consumer has been scheduled on this processor (1 longword).

To support the VECTOR class, MONITOR appends the records in Table B-9 to the system information record.

Table B-9 Descriptions of Additions to System Record Fields

Field	Symbolic Offset	Contents
VPCPUs	MNR_SYI\$B_VPCPUS	Number of vector-present processors in the current system (1 byte).
VP Conf	MNR_SYI\$L_VPCONF	Bit mask identifying those processors in the configuration that are vector-present processors (1 longword).

B.13 Network Control Program Utility (NCP)

This section describes new NCP line and circuit name support for VAXft 3000 systems and for two new Ethernet/820 controllers. See the *VMS Version 5.4 Release Notes* for more information about these and other hardware components that are new or enhanced for Version 5.4 of the VMS operating system.

B.13.1 Line and Circuit Name Support for VAXft 3000 Systems

The VMS Network Control Program Utility (NCP) supports the following new line and circuit name for VAXft 3000 systems (the controller number can be 0 or a positive number):

KFE-<controller number>

When you enter NCP commands from a VAXft 3000 system connected to your DECnet-VAX network, the KFE-*n* line and circuit name is displayed, as follows:

```
$ RUN SYS$SYSTEM:NCP
NCP> SHOW KNOW LINE
```

Line Volatile Summary as of 31-AUG-1990 12:50:03

Line	State
KFE-0	on

```
$ RUN SYS$SYSTEM:NCP
NCP> SHOW KNOW LINE
```

Circuit Volatile Summary as of 31-AUG-1990 12:52:03

Circuit	State	Loopback Name	Adjacent Routing Node
KFE-0	on		8.999 (JUPE)

B.13.2 Line and Circuit Names for New Ethernet/820 Controllers

The VMS Network Control Program Utility (NCP) now supports new line and circuit names for the following Ethernet/820 controllers. (See the *VMS Version 5.4 Release Notes* for a complete description of each new controller.)

- DEMNA controller—The NCP line and circuit name for the DEMNA controller is as follows:

MNA-<controller number>

For example:

MNA-0 (for EXAn)
MNA-1 (for EXBn)

- Second Generation Ethernet Controller (SGEC)—The NCP line and circuit name for the SGEC is as follows:

ISA-<controller number>

For example:

ISA-0 (for EZAn)
ISA-1 (for EZBn)

B.14 VMS Volume Shadowing Phase II

Volume shadowing is the process of maintaining multiple copies of the same data on two or more disk volumes. This duplication of data provides greater data availability and faster data accessibility. Volume shadowing provides high availability by ensuring against data loss resulting from media deterioration or through controller or device failure. When data is recorded on more than one disk volume, you have access to critical data even when one volume is unavailable. Disk input/output operations continue with the remaining members of the shadow set.

The system can also find data more quickly because it can search more than one disk. Because a shadow set is made up of multiple disks containing the same data, the shadow set can use the additional read heads to respond to multiple read requests at the same time. In addition, when normal media deterioration renders sections of a volume unreadable, systems with volume shadowing can read the duplicate data and copy it to the failing volume to repair data.

Before Version 5.4, the VMS operating system supported only phase I volume shadowing (see the *VAX Volume Shadowing Manual*). This type of shadowing provides centralized shadowing using HSC controllers with compatible DSA disks. Phase I shadowing is limited to CI configurations on a single system or a VAXcluster.

VMS Volume Shadowing phase II supports the following:

- Clusterwide shadowing of all MSCP-compliant DSA disks having the same physical geometry (having the same number of logical blocks) on a single system or located anywhere in a VAXcluster system.

Volume shadowing phase II supports clusterwide shadowing of all DSA devices. Phase II is not limited to HSC-controlled disks but extends volume shadowing capabilities to all DSA disks including local adapters, all DSSI (RF-series) disk devices on any VAX computer, all interfaces (including but not limited to the KFQSA interface), and across MSCP servers.

- Distributed, not centralized, shadowing

Volume shadowing phase II creates and maintains virtual units in a distributed fashion on each node in the cluster. Phase II supports shadowing on a single system or in a VAXcluster system where interprocessor communication is carried out over a computer interconnect (CI), Digital small systems interconnect (DSSI), mixed-interconnect configuration, or the Ethernet. Thus, volume shadowing provides fault tolerance resulting from disk media errors across the full range of VAX processors and configurations.

- Shadowing of the system disk and Files-11 On-Disk Structure Level 2 (ODS2) data disks.

- Shadowing capabilities across different controllers.

Shadow set member units can be located on different controllers and VMS MSCP servers.

- Shadowing capabilities with mixed phases.

It is possible to use both phase I and phase II shadowing on the same node at the same time. You can also mix phase I and phase II shadowing in a VAXcluster system.

See the new *VMS Volume Shadowing Manual* for complete information about phase II volume shadowing.

B.15 VMS Version 5.4 Programming Features

The following sections contain information about the VMS Version 5.4 programming features.

B.16 Larger Page Size Capability with Linker Utility

With Version 5.4 of the VMS operating system, you can now specify larger page sizes by using the new /BPAGE qualifier with the LINK command. Note that the /BPAGE qualifier affects only the construction of the image (shareable or executable), not the linker itself or any page-size dependencies in the linked program.

/BPAGE

Specifies the page size the linker should use when it creates the image sections that make up an image.

Format

/BPAGE [=page-size-indicator]

qualifier values**page-size-indicator**

Specify any of the values listed in the following table:

Value	Page Size	Defaults
9	512 bytes	Default value when the /BPAGE qualifier is not specified.
13	8 KB	Default value when the /BPAGE qualifier is specified without a value.
14	16 KB	-
15	32 KB	-
16	64 KB	-

Description

With Version 5.4 of the VMS operating system, you can specify which page size the linker uses to create an image by using the new /BPAGE qualifier with the LINK command.

The images the linker creates are made up of image sections that the linker allocates on page boundaries. When you specify a larger page size, the origin of image sections increases to the next multiple of that size.

The /BPAGE qualifier can be used with or without a value indicating the page size. When specified without a value, the linker creates image sections on 8KB page boundaries. To select another page size, assign the appropriate value from the table above. (The values represent the power of 2 that produce the page size desired. For example, to get an 8KB page size, specify the value 13 because 2^{13} equals 8K.) When the LINK command is used without the /BPAGE qualifier, the linker uses a page size of 512 bytes by default.

An image linked to a larger page size generally runs correctly on a current VMS system, but it might consume more virtual address space. In addition, linking a shareable image to a larger page size can cause the value of transfer vector offsets to change if they were not allocated in page 0 of the image. Do not link against a shareable image that was created with a different page size. (You cannot determine the page size used in the creation of a VAX image from the image.)

Example

```
$ LINK/BPAGE ALPHA.OBJ
```

When the /BPAGE qualifier is specified without a value, the linker creates image sections on 8KB page boundaries.

```
$ LINK ALPHA.OBJ
```

When /BPAGE is not specified, the linker uses 512-byte pages, by default.

```
$ LINK/BPAGE=16 ALPHA.OBJ
```

Including the value 16 with the /BPAGE qualifier causes the linker to create image sections on 64KB page boundaries.

B.17 VMS Record Management Services

This section describes the following enhancements to the VMS Record Management Services for Version 5.4 of the VMS operating system:

- Asynchronous support for process-permanent files
- Increase in local buffer limit
- Access-mode protection
- Expired-date suppression

B.17.1 VMS RMS Asynchronous Support for Process-Permanent Files

Prior to Version 5.4 of the VMS operating system, VMS RMS ignored the asynchronous option for process-permanent files. VMS RMS now supports this option, which affects the performance options within the following two RMS control blocks:

RMS Control Block	Field	Performance Option
File Access Block (FAB)	FAB\$L_FOP	FAB\$V_ASY
Record Access Block (RAB)	RAB\$L_ROP	RAB\$V_AST

B.17.2 Local Buffer Maximum Increased

With Version 5.4 of the VMS operating system, the maximum number of local buffers is increased to 32,767. Prior to Version 5.4, you were limited to specifying no more than 127 local buffers for a record stream from the VMS RMS interface using the RAB multibuffer count field (RAB\$B_MBF). You obtain the additional local buffering capability by using the **multibuffer count XABITM**. The multibuffer count XABITM is used as an input to the Connect service only. It is not used as an output by any service.

The maximum number of local buffers established by the DCL command SET RMS_DEFAULT for a *process* has also increased from 127 to 255. However, the maximum number of local buffers established by the DCL command SET RMS_DEFAULT for the *system* remains 127.

The XAB\$_MULTIBUFFER_COUNT XABITM requires a 4-byte buffer to store the value that specifies the number of local buffers. To specify the number of local buffers, set up the XAB\$_MULTIBUFFER_COUNT XABITM with the number of local buffers desired. Then, link the XABITM into the XAB chain for the record stream prior to invoking the Connect service. When you use the multibuffer count XABITM, the value specified overrides any value that resides in the RAB\$_MBF for the related record stream. See the *VMS Record Management Services Manual* for details about using a XABITM.

Before you increase the size of the local buffer pool, you should consider current memory management parameters because excessively large buffer pools introduce additional paging that can reduce I/O performance.

B.17.3 Access-Mode Protection for VMS RMS

VMS RMS now provides access-mode protection for its services and associated memory. This feature is analogous to the protection provided by the system services \$ASSIGN and \$SETPRT.

No code changes are required for RMS calls involving a single access mode. A code change might be required for RMS calls that initiate operations from an inner access mode and allow subsequent RMS operations from an outer access mode.

If an inner-mode caller initiates an RMS operation without overriding the access mode, subsequent outer-mode calls fail with an RMS\$_PRV error. The arguments in the following code example are used to override the caller's access mode. These arguments, together with related topics, are described in the *Introduction to VMS System Services*.

```
FAB$V_CHAN_MODE = PSL$C_<USER,SUPER,EXEC,KERNEL> ! Select one
```

VMS uses the maximized value of the caller's access mode and the FAB\$V_CHAN_MODE argument (RMS access-mode argument) to establish the access mode.

B.17.3.1 Access-Mode Protected Services

The following services initiate operations on files. These services establish the access mode that VMS RMS uses to validate the access modes of subsequent accessing services.

\$CREATE	\$OPEN	\$PARSE	\$SEARCH
----------	--------	---------	----------

The following services access open files to perform various VMS RMS operations. The access modes for each service trying to access an open file must be validated before RMS operations are allowed.

\$CLOSE	\$CONNECT	\$DELETE	\$DISCONNECT
\$DISPLAY	\$EXTEND	\$FIND	\$FREE
\$FLUSH	\$GET	\$NXTVOL	\$PUT
\$READ	\$RELEASE	\$REWIND	\$SPACE
\$TRUNCATE	\$UPDATE	\$WAIT	\$WRITE

VMS RMS does not validate the access mode for the following services because access-mode comparison is not relevant to them:

\$ENTER	\$ERASE	\$REMOVE	\$RENAME
---------	---------	----------	----------

B.17.3.2 Access-Mode Protected Memory

VMS RMS now protects the following data structures and their associated I/O buffers at EW (executive read/write). Previously, the data structures were protected at UREW (user read, executive write).

- RMS-controlled data structures
- Process-permanent data structures
- Image-activated data structures

The following memory protection exceptions apply to user-mode accessors of RMS and are protected at UREW:

- Internal RMS I/O buffers—to facilitate RAB\$V_LOC mode
- RMS buffers containing collated tables used for indexed files

B.17.4 Expired-Date Suppression

The file system, in conjunction with parameters established using the DCL interface (see the description of the SET VOLUME command in the *VMS DCL Dictionary*), gives users a facility for determining whether a data file has expired and is eligible to be transferred to another storage medium. Expiration of a file is determined by the Expiration Date and Time, which should not be updated for maintenance functions or for any function where the data is not really being modified.

Prior to VMS Version 5.4, the ability to suppress the expiration update was available only to applications that interface directly with the file system through the \$QIO system service. (See the *VMS I/O User's Reference Manual: Part I*.) Now the ability to selectively suppress the update of the Expiration Date and Time is available to all applications through the RMS interface.

B.17.4.1 The Role of XAB\$_NORECORD XABITM

The XAB\$_NORECORD XABITM suppresses the update of the Expiration Date and Time on the \$CLOSE service. The Expiration Date and Time is used by VMS to determine if the data in a disk file has been accessed recently. Normally, when data has been read or written to a disk file, the \$CLOSE service updates the Expiration Date and Time to the current date and time. This moves back the date and time when the file is considered expired. Specifying the XAB\$_NORECORD XABITM suppresses the update of the Expiration Date and Time.

The XAB\$_NORECORD XABITM uses a 4-byte buffer to set the NORECORD flag to logic 1 using the symbol XAB\$_ENABLE. Any other value in this XABITM buffer returns an RMS\$_XAB error. An application cannot disable this option because the Files-11 On-Disk Structure Level 2 ACP does not support disabling this function once it has been selected for an \$OPEN or \$CREATE service.

B.17.4.2 Applications for XAB\$_NORECORD XABITM

Typically, the XAB\$_NORECORD XABITM is used by directory or maintenance routines that do not manipulate the data and, therefore, do not change the expiration status of a disk file. For example, the DCL command DIRECTORY /FULL uses the XAB\$_NORECORD XABITM as it opens files to access prolog data containing key information. In this case, DIRECTORY displays prolog information but does not display or modify user data in the disk file and should not modify the Expiration Date and Time. Maintenance utilities should consider using this XABITM. For example, a disk defragmentation utility should not modify the expiration status of a disk file.

Digital recommends using the XAB\$_NORECORD XABITM on the \$OPEN service instead of on the \$CLOSE service—because the suppression of the Expiration Date and Time update is guaranteed should the file deaccess or should a close occur because of process deletion or RMS rundown.

XAB\$_NORECORD can be enabled on input to the \$CLOSE, \$OPEN, and \$CREATE services. If the \$CREATE service opens an existing file through the Create-if option and the Expiration Date and Time is not to be modified, the XAB\$_NORECORD XABITM can be specified. When the XAB\$_NORECORD XABITM is used on a \$CREATE service that *creates* a file, it disables the update on the subsequent \$CLOSE service but does not prevent initialization of the Expiration Date and Time on the file creation in the ACP.

The XAB\$_NORECORD XABITM can be sensed on output from RMS for the \$OPEN, \$CREATE, \$DISPLAY, and \$CLOSE services. An application typically senses the XAB\$_NORECORD XABITM to determine if the XABITM was specified on a previous \$OPEN or \$CREATE option or if it is specified by the current RMS operation.

B.18 System Dump Analyzer Utility (SDA)

This section describes two new qualifiers to the SHOW PROCESS command now available with Version 5.4 of the VMS System Dump Analyzer Utility (SDA).

B.18.1 New SHOW PROCESS Qualifier: /IMAGES

The /IMAGES qualifier to the SDA command SHOW PROCESS displays the address of the Image Control Block, the starting and end addresses of the image, the activation code, the protected and shareable flags, the image name, and the major and minor ID of the image.

The following is an example of output displayed by the SHOW PROCESS /IMAGES command:

```

                                Process activated images
                                -----
ICB          Start      End      Type          Image Name  Major ID,Minor ID
-----
7FF83878     00000200   00000DFF  MAIN          SHOW_PROC_IMAGES  0,0
7FF84100     0003AC00   0003FBFF  GLOBAL PRT SHR DECW$TRANSPORT_COMMON 12,12
7FF84400     00036200   0003ABFF  GLOBAL          CONVSHR  1,0
7FF84470     0002E400   000361FF  GLOBAL          FDLSHR  1,0
7FF84560     00021A00   0002E3FF  GLOBAL          SORTSHR  2,28
7FF845D0     00000E00   000089FF  GLOBAL          LIBRTL2  1,12
7FF835F8     00008A00   000219FF  GLOBAL SHR      LIBRTL  1,14
7FF84800     00060C00   000767FF  MERGED SHR      ADARTL  0,0
7FF84720     00076800   000A03FF  GLOBAL SHR      MTHRTL 129,32781
  
```

Total images = 9 Pages allocated = 1017

The following are possible values for the activation code:

- MAIN—Image is the object of a RUN command
- MERGED—Image is an additional mapped image
- GLOBAL—Image is a global image section

The protected flag (PRT) indicates that the image is installed protected. The shareable flag (SHR) indicates that the image is installed shareable.

For more information about the SDA command SHOW PROCESS, see the *VMS System Dump Analyzer Utility Manual*.

B.18.2 New SHOW PROCESS Qualifier: /VECTOR_REGISTERS

The System Dump Analyzer lets you examine vector instructions and vector context from a system dump file or in a running system. One way to accomplish this is by specifying the new /VECTOR_REGISTERS qualifier to the SHOW PROCESS command, which obtains the values of the registers from the process's vector context area. See Section B.2.3.5.2 for a complete description of SDA support for vector processing.

B.19 VMS RMS Journaling: Support for DECdtm Services

This section describes VMS RMS Journaling enhancements that support DECdtm services for Version 5.4 of the VMS operating system. (See Section B.3 for a complete description of DECdtm services.) VMS RMS Journaling continues to support existing applications developed on previous versions of VMS RMS Journaling.

B.19.1 Support for DECdtm Transactions

The DECdtm **transaction** has superseded the Recovery Unit Facility (RUF) **recovery unit**. In VMS RMS Journaling Version 5.4, an RMS recovery unit is the recoverable work performed by a single process within a DECdtm transaction.

The RUF recovery unit services have been superseded by corresponding DECdtm transaction services, as follows:

RUF Recovery Unit Service	DECdtm Transaction Service
\$START_RU	\$START_TRANS(W)
\$END_RU	\$END_TRANS(W)
\$ABORT_RU	\$ABORT_TRANS(W)

In addition, a single DECdtm transaction service, \$END_TRANS(W), has replaced two other RUF services, \$PREPARE_RU and \$COMMIT_RU, which together were equivalent to the \$END_RU service.

B.19.2 RUF Services Emulated

Recovery Unit Facility (RUF) services are still supported. They are emulated transparently using DECdtm transaction services.

You do not have to recompile or relink your applications to run them under VMS RMS Journaling Version 5.4.

You can convert an application that uses only one active transaction at a time to use the DECdtm services by replacing calls to RUF services with calls to the corresponding DECdtm transaction services.

However, combining DECdtm transaction services and RUF recovery unit services in a single image requires care. You should avoid having transactions that were started using the DECdtm services active at the same time as transactions that were started using the RUF services.

B.19.3 Network Support

Remote RMS files marked for recovery unit journaling can be modified within a transaction. They will be included in the *atomic unit of work* defined by the transaction. A **remote** file is a file accessed by a **client** RMS process through the DAP/FAL protocol to a "server" system.

The following conditions apply to remote files:

- Remote files can be marked for any combination of RU (recovery unit), AI (after-image), or BI (before-image) journaling.
- All journaling takes place locally with respect to each file.
- All recovery takes place locally with respect to each file.
- Both client and server nodes must support DECdtm (that is, must be running VMS Version 5.4 or later).

- The server node must be licensed for RMS Journaling.
- The DIRECTORY/FULL and ANALYZE/RMS commands have been enhanced to display the type of journaling enabled but not the names of any AI or BI journals.
- The SET FILE/AI_JOURNAL/BI_JOURNAL/RU_JOURNAL command can be applied to a locally accessed file only.

The following examples compare transactions using local or remote access:

Local Access

```
$OPEN file1
$CONNECT stream1 to file1
$OPEN file2
$CONNECT stream2 to file2
```

```
$START_TRANSW
$GET from stream1
$UPDATE to stream1
$PUT to stream2
$END_TRANSW
```

Remote Access

```
$OPEN file1
$CONNECT stream1 to file1
$OPEN n2::file2
$CONNECT stream2 to file2
```

```
$START_TRANSW
$GET from stream1
$UPDATE to stream1
$PUT to stream2
$END_TRANSW
```

The only difference between the two code examples is that, in the remote example, the second file specification includes a node name. As a result, RMS transparently manages two recovery units within the transaction.

The following table summarizes the differences between using recovery unit journaling locally and remotely:

Local Access	Remote Access
One transaction	One transaction
One recovery unit	Two recovery units
One RU journal	Two RU journals

B.19.4 Record Stream Association

In applications that use the DECdtm transaction services, an RMS record stream is associated with a transaction as a result of an RMS record operation. The application can use either the DECdtm default transaction or the new XABITM item list entry XAB\$_TID to determine which transaction the record stream should join.

B.19.4.1 How Streams Become Associated with a Transaction

Under RMS Journaling Version 5.4, record streams are associated with transactions as follows:

- If the DECdtm services are being used, then eligible streams associate with a transaction at the time of a record operation, not when the transaction is started or the stream is established (as was the case using RUF services).
- A record operation can cause stream association if its action is recoverable. The \$PUT, \$UPDATE, \$DELETE, \$FIND, \$FREE, \$GET, \$RELEASE, and \$REWIND services might cause an eligible stream to associate with a transaction.

VMS Version 5.4 Features

How Streams Become Associated with a Transaction

- A record operation must result in stream association if it affects record data in the file. The \$PUT, \$UPDATE, and \$DELETE services must cause an eligible stream to be associated with a transaction.

B.19.4.2 Stream Association Using RUF and DECdtm Services

The following example compares the way streams are associated with transactions under DECdtm and RUF:

Using DECdtm

```
$START_TRANSW
$GET from <parameter>(stream1)
$UPDATE to <parameter>(stream1)
```

Using RUF

```
$START_RU
$GET from <parameter>(stream1)
$UPDATE to <parameter>(stream1)
```

- Using Version 5.4 of RMS Journaling (DECdtm services), the stream associates on the \$GET service.
- Using RUF services with versions of RMS Journaling prior to 5.4 and emulation on Version 5.4, the stream associates on the \$START_RU.
- In most cases, this difference does not matter and a RUF application can be converted to the direct use of DECdtm services by simple substitution.
- In the cases where it does matter, the association at record operation time is more flexible than association at transaction start (using RUF).

B.19.5 Detached Recovery

The following sections describe modifications that have been made in the operation of detached recovery—specifically to the performance of synchronous, asynchronous, and partial recoveries.

B.19.5.1 Synchronous and Asynchronous Recovery

The RMS Detached Recovery server (new image SYS\$SYSTEM:RMSREC\$SERVER.EXE) can perform both synchronous and asynchronous recovery. Asynchronous recovery is the default mode; it proceeds as follows:

1. Detached recovery “adopts” orphaned transactions by acquiring the record locks for all records modified within a recovery unit. The detached recovery server is multithreaded and performs asynchronous system service calls (including RMS operations).
2. The detached recovery server indicates completion as soon as the record locks have been reacquired. Thus, access to records and files is reenabled sooner.
3. Actual recovery proceeds asynchronously with respect to the original request. This is in contrast to the synchronous recovery that was performed in versions of VMS RMS Journaling prior to Version 5.4.

Synchronous recovery is used in the following circumstances:

- Partial recovery—One or more secondary files are unavailable, so detached recovery cannot acquire all the record locks from an orphaned transaction. See Section B.19.5.2 for a detailed description of partial recovery.
- Limited resources—The detached recovery server does not have enough resources to acquire all the record locks on the file to be recovered (for example, a very large database with many active transactions).

- **Exclusive access**—The process that initiates detached recovery has tried to access the file such that it either has exclusive access to the file or it is the only process that can modify the file. (It may or may not allow shared read access.) In this case, the accessor will not look for record locks from other processes, and the locks owned by detached recovery can create difficulties for the accessor.

B.19.5.2 Partial Recovery

When detached recovery receives a request to recover a file, it tries to recover all the effects of all orphaned transactions that involve the file. The specific file for which RMS requests recovery is called the **primary** file. In addition to the changes made to the primary file, each of the orphaned transactions can also include changes to a number of other files. These additional files are called **secondary** files.

Recovery of secondary files is not required to allow access to the primary file. If detached recovery cannot access a secondary file referenced in a recovery unit journal for one of the orphaned transactions, then detached recovery cannot adopt that transaction. In such a case, detached recovery recovers that particular recovery unit journal in synchronous mode and omits all operations that involve the inaccessible secondary file. Omitting a secondary file is permissible, since it is necessary only to recover the primary file to satisfy the client's request. All the information necessary to recover the secondary file is left in the recovery unit journal for eventual use in recovering that file.

B.19.6 Placement of Recovery Unit Journals

In RMS Journaling Version 5.4, the location of a recovery unit journal is determined as follows:

- The first local stream that associates with the transaction selects the location for the RUJ file.
- By default, the recovery unit journal is on the same volume as the file.
- The SET FILE/RU_JOURNAL=(LABEL=volnam) command can specify a different volume for all accessors of the file.
- Each accessor can redirect the recovery unit journal by defining a different equivalence name for the logical DISK\$volnam.
- The XAB\$_RUJVOLNAM item-list entry on a XABITM block connected to the RAB can be used to override all the preceding factors.
- Recovery unit journals can be reused. When the transaction is completed, the recovery unit journal becomes idle.
- If the process does not have an idle recovery unit journal on the selected volume, then a new one is created.

The following example compares the placement of a recovery unit journal under DECdtm and RUF:

Using DECdtm

```
$START_TRANSW
$GET from parameter(stream1)
$UPDATE to parameter(stream1)
```

Using RUF

```
$START_RU
$GET fromparameter(stream1)
$UPDATE toparameter(stream1)
```

- Using VMS RMS Journaling Version 5.4 (DECdtm services), the recovery unit journal is created when the \$GET service is called.

VMS Version 5.4 Features

Placement of Recovery Unit Journals

- Using a version of VMS RMS Journaling prior to Version 5.4 (that is, RUF services), the recovery unit journal is created when the \$UPDATE service is called.
- Using RUF emulation on Version 5.4, the recovery unit journal is created when the \$START_RU service is called.
- With the VMS Version 5.4 operating system, even read-only transactions require a recovery unit journal, but it will not be written to.

B.19.7 Multiple Long-Term Journals Allowed

The files involved in a single transaction are no longer restricted to a single after-image journal and a single before-image journal.

B.19.8 Mixed-Version Clusters

Nodes using versions of VMS RMS Journaling prior to Version 5.4 of the VMS operating system can run together in a VAXcluster with nodes using Version 5.4. Shared access to files marked for journaling is supported in such a mixed-version cluster with one exception: you cannot use a node running an earlier version to recover a file that participated in a transaction that required a two-phase commit. VMS RMS Journaling Version 5.4 includes certain records ("prepare" records) in the journal that earlier versions do not understand.

The following examples show responses to three ways of trying to access the file [FINANCE]PAYROLL.DAT, which has a prepare record in its recovery unit journal, using a version of VMS RMS Journaling prior to Version 5.4:

- If your application tries to access the file directly, RMS returns the following error messages to your application:

```
$ TYPE PAYROLL.DAT
%TYPE-W-OPENIN, error opening WORK1:[FINANCE]PAYROLL.DAT;1 as input
-RMS-E-RRF, recovery unit recovery failed
-RMSREC-F-INVJNLFIL, invalid journal file
```

In addition, detached recovery sends the following messages to OPCOM:

```
***** OPCOM 30-MAY-1990 09:16:20.84 *****
Message from user BEETHOVEN on EROICA
%RMSREC-F-OPRHDRDET, error occurred during detached recovery unit recovery; init
iated by process ID (PID) 4A2004A0

***** OPCOM 30-MAY-1990 09:16:20.91 *****
Message from user BEETHOVEN on EROICA
%RMSREC-F-INVJNLFIL, invalid journal file

***** OPCOM 30-MAY-1990 09:16:20.92 *****
Message from user BEETHOVEN on EROICA
-RMSREC-F-JNLFILE, journal file DISK$WORK1:[SYSJNL]RMS$0000001E.RMS$JOURNAL;24

***** OPCOM 30-MAY-1990 09:16:20.93 *****
Message from user BEETHOVEN on EROICA
-RMSREC-F-INVJNLIDX, invalid journal index number
```

- If you try to use the Recover Utility (RECOVER) on the file, RECOVER responds with the following messages:

```
%RMSREC-F-NOTCOMREC, file was not completely recovered as requested
%RMSREC-F-LSTVALTIM, time of last valid record: 28-MAY-1990 13:18:06.27
%RMSREC-F-INVJNLFIL, invalid journal file
-RMSREC-F-JNLFILE, journal file DISK$WORK1:[FINANCE]PAYROLL.AIJ1;1
-RMSREC-F-CURNOTSUPP, journal entry: 12 currently not supported
```


- If the file is being accessed by a process on a node running a version of the VMS operating system prior to Version 5.4 and by a process on a Version 5.4 node and the Version 5.4 node fails, the surviving accessor on the other node attempts to perform detached recovery. Detached recovery fails, deletes the surviving process, and sends the following messages to OPCOM:

```
***** OPCOM 30-MAY-1990 09:16:20.84 *****
Message from user BEETHOVEN on EROICA
%RMSREC-F-OPRHDRDET, error occurred during detached recovery unit recovery; init
iated by process ID (PID) 4A2004A0

***** OPCOM 30-MAY-1990 09:16:20.91 *****
Message from user BEETHOVEN on EROICA
%RMSREC-F-INVJNLFIL, invalid journal file

***** OPCOM 30-MAY-1990 09:16:20.92 *****
Message from user BEETHOVEN on EROICA
-RMSREC-F-JNLFILE, journal file DISK$WORK1:[SYSJNL]RMS$0000001E.RMS$JOURNAL;24

***** OPCOM 30-MAY-1990 09:16:20.93 *****
Message from user BEETHOVEN on EROICA
-RMSREC-F-INVJNLIDX, invalid journal index number
```

To recover the file, you must perform recovery on, or access the file from, a node running VMS RMS Journaling Version 5.4, or you must upgrade the remaining nodes in your VAXcluster to Version 5.4 of the VMS operating system.

The following information is provided for your information only. It is not intended to be used as a substitute for professional advice. The information is provided as a service to our customers and is not intended to be used as a substitute for professional advice. The information is provided as a service to our customers and is not intended to be used as a substitute for professional advice.

The following information is provided for your information only. It is not intended to be used as a substitute for professional advice. The information is provided as a service to our customers and is not intended to be used as a substitute for professional advice. The information is provided as a service to our customers and is not intended to be used as a substitute for professional advice.

The following information is provided for your information only. It is not intended to be used as a substitute for professional advice. The information is provided as a service to our customers and is not intended to be used as a substitute for professional advice. The information is provided as a service to our customers and is not intended to be used as a substitute for professional advice.

VMS Version 5.3 Features

This appendix describes features that were new to Version 5.3 of the VMS operating system but are not yet documented in other printed manuals.

C.1 VMS Version 5.3 System Management Features

This section describes enhancements to the following components of the VMS operating system:

- Lock Manager
- NCP Executor Commands

C.1.1 Extension of Lock Manager Limit

The Lock ID space for the Lock Manager is now extended from 65,535 to 262,144 locks. The SYSGEN parameters listed in the following table are increased to the values indicated:

SYSGEN Parameter	New Maximum Value
LOCKIDTBL	262,144
LOCKIDTBL_MAX	262,144
SRPCOUNT	270,336
SRPCOUNTV	270,336
IRPCOUNT	135,168
IRPCOUNTV	135,168

C.1.2 NCP Executor Command Changes

The NCP executor commands now include the following:

- A new parameter to SET/DEFINE EXECUTOR command
- New display characteristics for SHOW EXECUTOR CHARACTERISTICS command

C.1.3 Parameter for SET/DEFINE EXECUTOR

The network ancillary control process (NETACP) manages an index into a properly synchronized table in nonpaged-pool memory. System managers can modify the size of the table using the NCP command SET/DEFINE EXECUTOR with the new parameter MAXIMUM DECLARED OBJECTS.

VMS Version 5.3 Features

Parameter for SET/DEFINE EXECUTOR

Parameter	Description
MAXIMUM DECLARED OBJECTS	Specifies the number of objects that processes can declare. To determine the current number of declared objects on your system, use the NCP SHOW KNOWN OBJECTS command. Each of the objects with a process identification (PID) listed is one declared object. A single process can declare more than one object. Failure to provide a sufficient number of objects can result in the failure of network servers to be initialized. The default of 31 objects is sufficient for most configurations. The valid range is 8 to 16383. Note that dynamically setting the number lower has no effect.

C.1.4 SHOW EXECUTOR CHARACTERISTICS Command

The SHOW EXECUTOR CHARACTERISTICS command now displays information as shown in the following example. Note that a new entry, Maximum Declared Objects, is displayed and the Pipeline quota now shows 10000.

```
NCP> SHOW EXECUTOR CHARACTERISTICS
```

```
Node Volatile Characteristics as of 16-JUN-1990 10:48:27
```

```
Executor node = 2.11 (BOSTON)
```

```

Identification      = DECnet-VAX V5.3,  VMS V5.3
Management version  = V4.0.0
Incoming timer      = 45
Outgoing timer      = 45
Incoming Proxy      = Enabled
Outgoing Proxy      = Enabled
NSP version         = V4.1.0
Maximum links       = 128
Delay factor        = 80
Delay weight        = 5
Inactivity timer    = 60
Retransmit factor   = 10
Routing version     = V2.0.0
Type                = routing IV
Routing timer       = 600
Broadcast routing timer = 40
Maximum address     = 1023
Maximum circuits    = 16
Maximum cost        = 1022
Maximum hops        = 15
Maximum visits      = 63
Maximum area        = 63
Max broadcast nonrouters = 64
Max broadcast routers = 32
Maximum path splits = 1
Area maximum cost   = 1022
Area maximum hops   = 30
Maximum buffers     = 100
Buffer size         = 576
Default access      = incoming and outgoing
Pipeline quota      = 10000
Alias incoming      = Enabled
Alias maximum links = 32
Alias node          = 2.10 (CLUSTR)
Path split policy   = Normal
Maximum Declared Objects = 31

```


C.2 VMS Version 5.3 Support for the VMS Distributed Name Service

The Distributed Name Service (DNS) is a facility for storing the names of resources in your network such as files, disks, nodes, queues, and mailboxes. The Distributed Name Service clerk is the VMS programming interface to DNS that allows an application to register a resource in the name service and then access the resource from any point in the network by a single name. DNS is a layered product and must be installed in your network before you can start the DNS clerk or utilize the name service.

Applications that need the Distributed Name Service must use the \$DNS clerk system service and the DNS run-time routines to register, modify, and locate information in the DNS database. A DNS clerk, which is resident on every VMS Version 5.3 or later system, receives application requests through the \$DNS system service. The clerk locates a DNS server that can process the request. Once the request is satisfied, the clerk returns the requested information to the client application.

The information in this section is intended for VMS programmers who are writing applications that call the Distributed Name Service. It includes the following:

- Conceptual information on DNS
- DNS clerk system services, \$DNS and \$DNSW
- DNS run-time routines
- Startup information for the DNS clerk
- DECnet event messages from the DNS clerk

See the *VMS System Messages and Recovery Procedures Reference Manual* for information about system error messages generated by the DNS clerk.

C.2.1 Introduction to the Distributed Name Service

The VAX Distributed Name Service (DNS) provides a means of assigning unique names to network resources so that a network application or network user can find resources within the network. (Resources are such things as disks, systems, applications, and so on.) Once an application has named a resource using DNS, the name is available for all users of the application. Multiple users located throughout a network can refer to a common resource by the same name. Resources can be moved within the network. No additional preparation is required, and it is not necessary to learn a new naming convention.

You should consider using DNS applications that need to access such remote resources as printers, files, disks, and nodes. In addition, application databases or servers are good candidates for naming. All of these resources would be commonly named and their locations identified within DNS. With DNS, the resource could be moved without users being aware of the change.

Although it is desirable to name application databases, you should ordinarily use DNS to store only the location of the database, not the database itself. (Most database applications require higher levels of consistency than DNS provides.) If the database is relocated, then only the DNS information has to be modified.

C.2.2 The DNS Namespace

The collection of names in the Distributed Name Service database is called a **namespace**. A namespace is located on VMS nodes where the DNS server software is installed. The collection of databases stored on each server makes up the namespace itself.

DNS refers to the named resources in a namespace as **objects**. Each object name refers to a specific entity. The object name is important because applications use the object name in all DNS operations.

Associated with every object is a set of **attributes** describing properties of an object. An application reads object attributes for information such as an address, class, or version.

Most applications use the address attribute of an object, which allows you to find the node on which a resource resides. When a network resource is relocated, an application has DNS update the object's address attribute. All requests for the object receive the new address. Since the object has the same DNS name, the application user can be unaware that the resource has moved.

C.2.2.1 Planning Namespace Objects

When writing applications that use DNS, it is important to determine ahead of time what resources an application needs and how an application will use each resource. Then you can determine what objects an application needs to create and the kind of information each object needs to store. Once the object is designed, you can decide which object attributes to assign and what their values will be.

C.2.2.2 Restrictions

Because of the high cost of keeping copies of DNS names synchronized, you should use DNS applications that store information that does not change frequently. Frequent updates add traffic to the network, which can degrade overall network performance. Because resources such as files, disks, nodes, queues, and mailboxes remain on one node for a long time, a good example of information to store with DNS is a network address.

Not only should the information stored in DNS be relatively static, it should also be verifiable. When DNS updates its database, it attempts to send the update to all copies of the name within 24 hours. This means that your application can request data from a copy of a name that has not been updated. An application must be able to recognize when data is invalid. For this reason, a network address is a good example of data that can be validated. If you use an address and the resource is not there, the data is obviously outdated.

C.2.2.3 Using the Namespace

An application choosing to use the namespace performs four basic operations:

- Object creation—An application needs to create an object to represent each network resource it requires.
- Object modification—Once an object is created to represent a resource, an application modifies the object to contain the attributes and values the application requires.
- Object deletion—When a resource is no longer useful, an application should delete the object.
- Information retrieval—The most common operation an application performs is requesting the DNS clerk to obtain the values of an attribute so that, for example, the application can locate the resource in the network.

C.2.2.4 Object Names

The name DNS assigns to an object is one that the user supplies. The client application translates the name it receives through the user interface from string format into **opaque** format before passing it to the DNS clerk. DNS works only with opaque format because it is guaranteed to be unique, whereas string format often contains logical names that easily change.

The \$DNS system service supplies functions for conversion between string and opaque format. If an application maintains its own databases, then the application must store DNS names in opaque format.

C.2.2.5 Object Attributes

Client applications store information about a resource as object attributes. When creating an object, an application needs to assign a class name and a version to a new object. The class name reflects the purpose of the object within an application. The purpose can be specific to an application or it can be shared among a group of applications. For example, a group of user names might be shared. An application uses the class name to search for its objects or list its objects. The class version helps to pair a version of an object with a software version.

To store additional information with an object, an application must modify the object.

DNS always assigns certain attributes to an object during creation. It assigns a unique identifier (UID) and an update time-stamping (UTS) indicating when an object was last edited. DNS also assigns a third attribute that specifies access control for the new object. Initially, the owner of the object has read, write, delete, control, and test access. The namespace administrator can modify this access according to site requirements.

An attribute name is limited to 31 characters and its value cannot exceed 4000 bytes. The name service assigns a prefix of DNS\$ to the name of each attribute it assigns. An application creates a prefix to assign to attributes it creates. For example, DECnet uses the prefix DNA\$ and the Distributed File Service uses the prefix DFS\$. Names assigned by Digital all contain the dollar sign (\$). User-supplied names should use an underscore (_). To ensure uniqueness, you should register your facility name through Digital's product registration program.

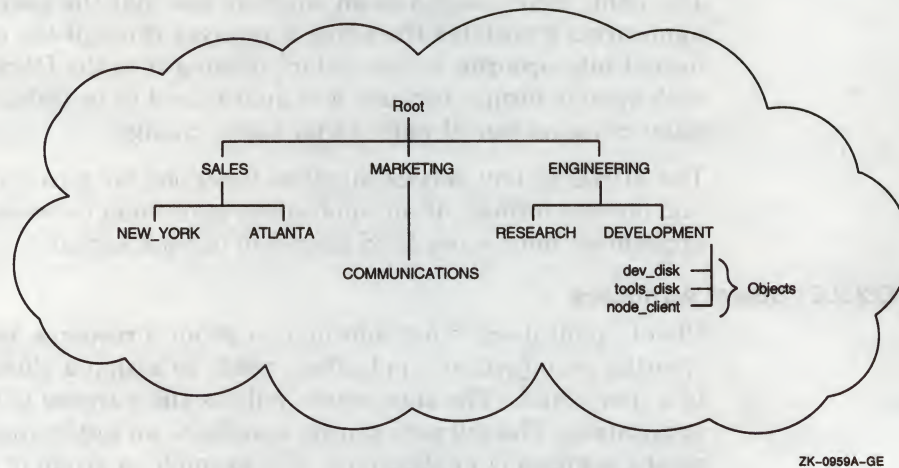
C.2.3 Structure of a Namespace

A DNS namespace is a hierarchical set of directories, as depicted in Figure C-1. At the top of the hierarchy is the root directory, which is symbolized by a period (.). Below the root directory are levels of subdirectories. The namespace administrator establishes the directory structure of the namespace and, in some cases, assigns names to directories. While the organization of the namespace directories is similar to the VMS directory structure, namespace directories are completely separate from the VMS directory structure.

VMS Version 5.3 Features

Structure of a Namespace

Figure C-1 DNS Namespace



Directories in a namespace can contain three types of entries:

- Objects
- Directory pointers
- Soft links

An **object** represents a network resource. It consists of a name that is unique within the namespace and its associated attributes.

Directory pointers are used internally by DNS to link one level of directories to the next. DNS refers to the hierarchical relationship of directories in terms of **child directories** and **parent directories**.

A **soft link** provides an alternate name for an object, directory, or soft link. For example, a namespace structured with both an organizational and a geographical dimension might access a single object through multiple soft links. A soft link can also be useful in renaming an object. The soft link would point to the original object name so that users could successfully use an outdated name. This kind of soft link would be deleted after sufficient time has passed for applications and users to become aware of the new object name. You create and delete links through the DNS management program.

Although an application requests the creation of an object in order to register a resource, it does not position the object in the namespace. The system administrator determines which directory DNS stores the object in. The structure of a namespace differs for each network, so you should not hard-code names into applications.

C.2.3.1 Naming Syntax

The DNS name of an object, directory, or soft link in the namespace is a complete path specification from the root directory to the entity in the directory of interest. For example, the DNS name `.ENGINEERING.DEVELOPMENT.TOOLS_DISK` identifies an object named `TOOLS_DISK` in the namespace directory called `.ENGINEERING.DEVELOPMENT`. The `ENGINEERING` directory is in the root directory, and `DEVELOPMENT` is a child directory of the `ENGINEERING` directory.

While the full name is a complete path name from the root directory, each element in a full name is called a **simple name**. The last simple name in a full name designates an object, a child directory, or a soft link. In the previous example, `TOOLS_DISK` is a simple name assigned to a disk object. The maximum length of a simple name is 255 bytes.

You can represent a full name in three ways:

`namespaceName:simpleName.simpleName`

or

`.simpleName.simpleName`

or

`simpleName.simpleName`

If the full name does not start with a namespace name or a period, DNS attempts to translate the first simple name as a logical name. Any equivalence name found is added to the name string in place of the matched simple name. This process is repeated until the first term does not match a logical name or the clerk encounters either a namespace name or a leading period. (A namespace name, assigned during DNS server installation, defaults to `node-name_NS`.)

The following example shows what happens with the name `RESEARCH.PROJECT_DISK`:

1. Look up `RESEARCH` as a logical name.
`RESEARCH` translates to `ENG.RESEARCH`, so the name string expands to `ENG.RESEARCH.PROJECT_DISK`.
2. Look up `ENG` as a logical name.
`ENG` translates to `.ENGINEERING`, so the name string becomes `.ENGINEERING.RESEARCH.PROJECT_DISK`. Because the new name has a leading period, translation stops.
3. The namespace name, `INMAX_NS`, is added to the front of `.ENGINEERING` because it is not explicitly specified. (A namespace administrator establishes the namespace name during installation.) The name becomes `INMAX_NS:.ENGINEERING.RESEARCH.PROJECT_DISK`.

C.2.3.2 Logical Names

When the DNS clerk is started on a VMS operating system (see Section C.2.10), the VMS system creates a unique logical name table for DNS to use in translating full names. This logical name table, called `DNS$SYSTEM`, prevents unintended interaction with other system logical names. The DNS use of logical names in parsing full names is described in Section C.2.3.1.

To define systemwide logical names for DNS objects, use the DCL command `DEFINE`. For example, to create the logical `RESEARCH.PROJECT_DISK` shown in the previous section, you would enter the following DCL command:

```
$ DEFINE/TABLE=DNS$SYSTEM RESEARCH "ENG.RESEARCH"
```

When parsing a name, the `$DNS` service specifies the logical name `DNS$LOGICAL` as the table it uses to translate a simple name into a full name. This name ordinarily translates to `DNS$SYSTEM` in order to access the systemwide DNS logical name table.

VMS Version 5.3 Features

Logical Names

To define process or job logical names for \$DNS, you must create a process or job table and redefine DNS\$LOGICAL as a search list, as in the following example (note that elevated privileges are required to create a job table):

```
$ CREATE /NAME_TABLE DNS_PROCESS_TABLE
$ DEFINE /TABLE=LNM$PROCESS_DIRECTORY DNS$LOGICAL -
_$ DNS_PROCESS_TABLE,DNS$SYSTEM
```

Once you have created the process or job table and redefined DNS\$LOGICAL, you can create job-specific logical names for DNS using the DCL command DEFINE, as follows:

```
$ DEFINE /TABLE=DNS_PROCESS_TABLE RESEARCH "ENG.RESEARCH.MYGROUP"
```

For information about logical names, see the *Introduction to VMS System Services*.

C.2.3.3 Valid Characters for DNS Names

DNS namespace names, full names, or simple names can contain letters, numbers, and certain punctuation marks from the ISO Latin 1 character set, as shown in Figure C-2. Additional characters and punctuation marks can appear as long as the name is enclosed in quotation marks, for example, "project%". See Figure C-3.

Figure C-2 Valid Character Codes for DNS Simple Names

Code Range (Decimal)	Character
036	\$
045	-
048-057	0 1 2 3 4 5 6 7 8 9
065-077	A B C D E F G H I J K L M
078-090	N O P Q R S T U V W X Y Z
095	-
097-109	a b c d e f g h i j k l m
110-122	n o p q r s t u v w x y z
192-207	À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
208-214	Ð Ñ Ò Ó Ô Õ Ö
216-223	Ø Ù Ú Û Ü Ý Þ ß
224-239	à á â ã ä å æ ç è é ê ë ì í î ï
240-246	ð ñ ò ó ô õ ö
248-255	ø ù ú û ü ý þ ÿ

ZK-0961A-GE

Note

All simple names containing the dollar sign (\$) are reserved for use by Digital.

Figure C-3 Additional Character Codes Allowed in Quoted Simple Names

Code Range (Decimal)	Character
032-033	{space} !
035	#
037-044	% & ' () * + ,
046-047	. /
058-064	: ; < = > ? @
091-094	[\] ^
096	'
123-126	{ } ~
160-167	{no-break space} ¡ ¢ £ ¤ ¥ ¦ §
168-174	¨ © ª « ¬ ® ¯
175-187	° ± ² ³ ´ µ ¶ · ¸ ¹ º »
188-191	¼ ½ ¾ ¿
215	X
247	÷

ZK-0962A-GE

DNS maintains the case of an entity when it registers an object, but it is case insensitive in lookups. For example, the name eng.research would match the name ENG.RESEARCH.

DNS also supports binary simple names. A binary name consists of the leading character pair %x or %X, followed by pairs of hexadecimal digits. A binary simple name does not match any regular or quoted simple name, even if a given name has the same binary value.

DNS makes use of wildcards for identifying groups of objects during search operations. Wildcards consist of the following:

Symbol	Name	Meaning
?	Question mark	Match one character.
*	Asterisk	Match any number of characters.

C.2.4 Creating Objects

Each application that uses DNS must register its resources in the namespace using either the \$DNS or the \$DNSW system service. Registration involves creating an object in the namespace to represent the resource. You create an object to represent each resource in the network that your application needs to find. At the same time, you should define attributes the object needs and assign their values.

A DNS object consists of a name and its associated attributes. You create the object first, along with some key attributes. Later, you can modify the object to hold additional attributes that are relevant to the application.

To create an object with \$DNS:

1. Prompt for a name from the user interface.

The name that an application assigns to an object should come from a user interface, a configuration file, a system logical, or some other source. The application never assigns an object's name because the namespace structure is uncertain. The name the application receives from the user interface is in string format.

VMS Version 5.3 Features

Creating Objects

2. Use the \$DNS parse function to convert the full name string into the opaque format of DNS.
3. Optionally, reserve an event flag so you can check for completion of the service.
4. Build an item list containing the following elements:
 - The opaque name for the object (resulting from the translation in step 2)
 - The class name given by the application, which should contain the facility code
 - The class version assigned by the application
 - An optional timeout value, specifying when the call expires
5. Optionally, provide the address of the DNS status block to receive status information from the name service.
6. Optionally, provide the address of the asynchronous system trap (AST) service routine. AST routines allow a program to continue execution while waiting for parts of the program to complete.
7. Optionally, supply a parameter to pass to the AST routine.
8. Call the create object function, providing all the parameters supplied in steps 1 through 7.

If a clerk call is not complete when timeout occurs, then the call completes with an error. The error is returned in the DNS status block.

An application should check errors returned; it is not enough to check the return of the \$DNS call itself. You need to check the DNS status block to be sure there are no errors at the DNS server.

The following C program shows how to create an object in the namespace with the synchronous service \$DNSW. The routine demonstrates how to construct an item list.

```
#include <dnsdef.h>
#include <dnsmsg.h>
/*
 * Parameters:
 *   class_name = address of the opaque simple name of the class
 *               to assign to the object
 *   class_len  = length (in bytes) of the class opaque simple name
 *   object_name= address of opaque full name of the object
 *               to create in the namespace.
 *   object_len = length (in bytes) of the opaque full name of the
 *               object to create
 */
create_object(class_name, class_len, object_name, object_len)
unsigned char *class_name;
unsigned short class_len;
unsigned char *object_name;
unsigned short object_len;
{
    struct $dnsitmdef createitem[4]; /* Item list used by system service */
    struct $dnscvrsdef version;      /* Version assigned to the object */
    struct $dnst iosb;               /* Used to determine DNS server status */
    int status;                      /* Status return from system service */
}
```



```

/*
 * Construct the item list that creates the object:
 */
createitem[0].dns$w_itm_size = class_len; ①
createitem[0].dns$w_itm_code = dns$_class;
createitem[0].dns$a_itm_address = class_name;

createitem[1].dns$w_itm_size = object_len; ②
createitem[1].dns$w_itm_code = dns$_objectname;
createitem[1].dns$a_itm_address = object_name;

version.dns$b_c_major = 1; ③
version.dns$b_c_minor = 0;

createitem[2].dns$w_itm_size = sizeof(struct $dnscversdef); ④
createitem[2].dns$w_itm_code = dns$_version;
createitem[2].dns$a_itm_address = &version;

*((int *)&createitem[3]) = 0; ⑤

status = sys$dnsnw(0, dns$_create_object, &createitem, &iosb, 0, 0); ⑥
if(status == SS$_NORMAL)
{
    status = iosb.dns$l_dnsb_status; ⑦
}

return(status);
}

```

The following list explains how the C program constructs an item list:

- ① The first entry in the item list is the address of the opaque simple name representing the class of the object.
- ② The second entry in the item list is the address of the opaque full name for the object.
- ③ The next step is to build a version structure, which will indicate the version of the object. In this case, the object is version 1.0.
- ④ The third entry in the item list is the address of the version structure that was just built.
- ⑤ Zero terminates an item list.
- ⑥ Call the system service to create the object.
- ⑦ Check to see that both the system service and DNS were able to perform the operation without error.

C.2.5 Modifying Objects

After applications use DNS to create objects that identify resources, they add attributes to the newly created objects that describe properties of the object.

You modify an object whenever you need to add an attribute, change an attribute value, or delete an attribute. You can add as many attributes as you like. If you add the same attribute to an object twice, the time-stamping on the attribute is updated.

DNS attributes can have a single value or they can have a set of values. For example, an attribute holding the class version number of a resource would have a single value, while an attribute holding the location of a service in the network could have a set of values. The set would hold the addresses of all nodes in the network that offer the service. Depending on the attribute type, DNS performs a slightly different action. DNS adds or deletes a value when there is only one.

VMS Version 5.3 Features

Modifying Objects

When there is a set of values, DNS adds or deletes a value from an existing group of values.

To modify an object with \$DNS:

1. Build an item list containing the following elements:
 - The opaque name of the object you are modifying
 - The type of entry, as described in Section C.2.3
 - The operation to perform
 - The type of attribute you are adding—a single value or a set of values
 - The attribute name
 - The value being added to the attribute
2. Supply any of the optional parameters described in Section C.2.4.
3. Call the modify attribute function, supplying the parameters established in steps 1 and 2.

The following C program shows how to add an attribute and its value to an object:

```
#include <dnsdef.h>
#include <dnsmsg.h>
/*
 * Parameters:
 *   obj_name = address of opaque full name of object
 *   obj_len  = length of opaque full name of object
 *   att_name = address of opaque simple name of attribute to create
 *   att_len  = length of opaque simple name of attribute
 *   att_value= value to associate with the attribute
 *   val_len  = length of added value (in bytes)
 */
add_attribute(obj_name, obj_len, att_name, att_len, att_value, val_len)
unsigned char *obj_name;
unsigned short obj_len;
unsigned char *att_name;
unsigned short att_len;
unsigned char *att_value;
unsigned short val_len;
{
    struct $dnsitmdf moditem[7]; /* Item list for $DNSW */
    unsigned char objtype = dns$k_object; /* Using object entries */
    unsigned char opertype = dns$k_present; /* Adding an object */
    unsigned char atttype = dns$k_set; /* Attribute will be type set */
    struct $dnsb iosb; /* Used to determine DNS status */
    int status; /* Status of system service */

    /*
     * Construct the item list to add an attribute to an object.
     */
    moditem[0].dns$w_itm_size = obj_len;
    moditem[0].dns$w_itm_code = dns$_entry;
    moditem[0].dns$a_itm_address = obj_name; ①

    moditem[1].dns$w_itm_size = sizeof(char);
    moditem[1].dns$w_itm_code = dns$_lookingfor;
    moditem[1].dns$a_itm_address = &objtype; ②

    moditem[2].dns$w_itm_size = sizeof(char);
    moditem[2].dns$w_itm_code = dns$_modoperation;
    moditem[2].dns$a_itm_address = &opertype; ③
```



```

moditem[3].dns$w_itm_size = sizeof(char);
moditem[3].dns$w_itm_code = dns$_attributetype;
moditem[3].dns$a_itm_address = &atttype; ④

moditem[4].dns$w_itm_size = att_len;
moditem[4].dns$w_itm_code = dns$_attributename;
moditem[4].dns$a_itm_address = att_name; ⑤

moditem[5].dns$w_itm_size = val_len;
moditem[5].dns$w_itm_code = dns$_modvalue;
moditem[5].dns$a_itm_address = att_value; ⑥

*((int *)&moditem[6]) = 0; ⑦

/*
 * Call $DNSW to add the attribute to the object.
 */
status = sys$dns(0, dns$_modify_attribute, &moditem, &iosb, 0, 0);
if(status == SS$_NORMAL)
{
    status = iosb.dns$l_dnsb_status;
}

return(status);
}

```

The following list explains how the C program adds an attribute and its value to an object:

- ① The first entry in the item list is the address of the opaque full name of the object.
- ② The second entry in the item list shows that the entry is an object—not a soft link or directory pointer.
- ③ The third entry in the item list is the operation to perform. The program adds an attribute with its value to the object.
- ④ The fourth entry in the item list is the attribute type. The attribute has a set of values rather than a single value.
- ⑤ The fifth entry in the item list is the opaque simple name of the attribute being added.
- ⑥ The sixth entry in the item list is the value associated with the attribute.
- ⑦ Check to see that both the system service and DNS performed the operation without error.

C.2.6 Distributing the Namespace

A VMS node running DNS server software can contain the entire namespace. However, performance and reliability are enhanced when several VMS nodes act as DNS servers.

DNS supports the **partitioning** of the namespace across several DNS servers. In this situation, no DNS server contains the entire namespace, but each contains a portion of the namespace, usually the directories frequently accessed by local client applications. Directory pointers connect parts of the database that are distributed among two or more servers.

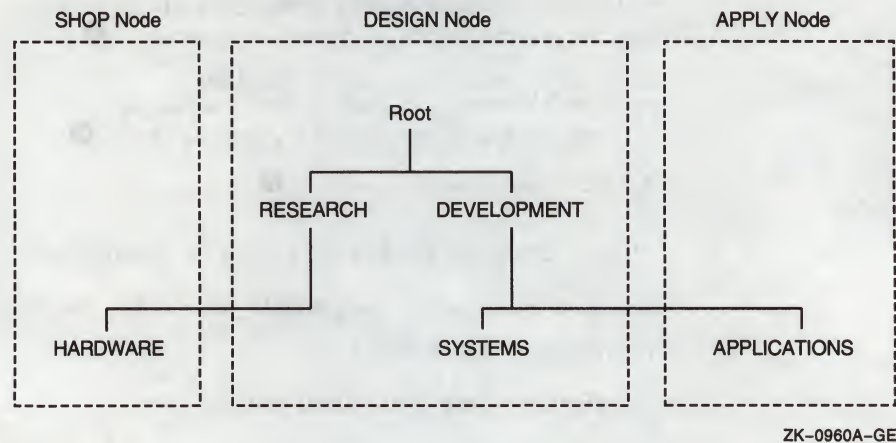
Figure C-4 depicts a namespace with three DNS servers. The DESIGN node contains most of the namespace—the root directory plus the research and development directories. The applications directory resides on the APPLY node, while the hardware directory resides on the SHOP node.

VMS Version 5.3 Features

Distributing the Namespace

DNS refers to a collection of directories on a server as a **clearinghouse**.

Figure C-4 Partitioned Namespace



C.2.6.1 Replicating Directories

In large networks, many applications rely on DNS and names must be available for the application to work. To ensure availability, DNS allows the duplication of data and provides a mechanism to keep all copies of names synchronized. Then, if one server becomes disabled, applications can still access the namespace through another server. Whenever data is duplicated, DNS copies one or more directories with all their contents.

The namespace administrator determines how many copies of each directory should exist and where they should be located. For example, Figure C-5 shows the same namespace as Figure C-4. However, in Figure C-5 the root directory is duplicated so that it exists on all three DNS servers.

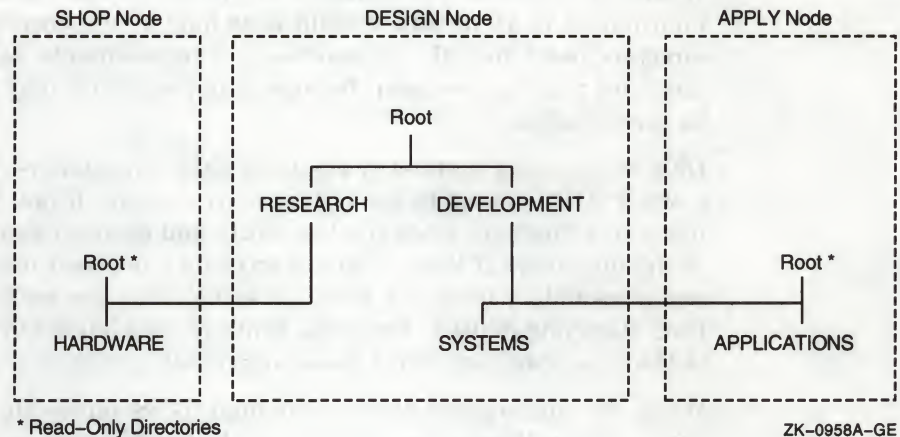
C.2.6.2 Types of Directories

Once you duplicate parts of a namespace, you generate different types of directories. Some are writable, while others are read-only. In a replicated namespace, there are three types of directories:

- Master
- Secondary
- Read-only

For example, in Figure C-5 there are three copies of the root directory. The master copy resides on node DESIGN. Read-only copies reside on the other two nodes.

Figure C-5 Namespace with Replicated Directories



In a master directory, an application can create or modify all types of entries: objects, directory pointers, and soft links. In a secondary directory, an application can create or modify objects and soft links but not directory pointers. An application can retrieve namespace data from any type of directory.

When an application attempts to create a new object or update an existing one, the DNS clerk sends the request to a DNS server that has a secondary or master directory. The request to create an object succeeds as long as no other entry with the same name exists; the request to modify an object succeeds as long as the object is found in the directory.

C.2.6.3 Setting Confidence

An application can use the confidence argument in a \$DNS call to stipulate the type of directory that the DNS clerk should use to service the call. For example, when an application wants to create an object, it can force the DNS clerk to create the object in the master directory by stipulating a high confidence level. Otherwise, DNS creates the object either in the master directory or in a secondary directory.

In a create or modify call, confidence has the following meaning:

- High confidence—Use the master directory.
- Medium confidence—Use the master directory or a secondary directory. There can be multiple copies of secondary directories.

An application's expression of confidence has a slightly different meaning in a request to find data. In this operation, there are three levels of confidence:

- High confidence—Use the master directory.
- Medium confidence—Use cached information to find the location of a DNS server but get the information from a DNS server.
- Low confidence—Use cached information.

VMS Version 5.3 Features

Maintaining Consistency in Data

C.2.6.4 Maintaining Consistency in Data

Whenever a directory is modified, the name service attempts to send the updated information to all directory replicas as long as the **convergence** attribute of the directory is set to high. Sometimes it is impossible to deliver the updates to all directory replicas, however, because a network link may be down or a node may be unreachable.

DNS does have a method of ensuring data consistency—it is called a **skulk**. In a skulk, DNS checks to see if data is consistent. If not, it gathers all updates made to a directory since the last skulk and propagates the updates to all replicas of the directory. If there is any discrepancy between information in a master and a secondary directory during a skulk, then the entry with the most recent time-stamping is used. Once the skulk is completed, DNS informs all directories of the time-stamping of the latest universal update.

When the convergence attribute is high, DNS skulks the namespace every 12 hours. When the convergence is low, the skulk occurs every 24 hours.

Directory replicas can lose their consistency between skulks. Two objects of the same name could be created simultaneously in different directory replicas or updates to the namespace might not be seen by all copies immediately. When DNS detects a conflict in replicas, it preserves the object with the most recent update time-stamping and deletes the older object. There is a chance that an application may get information from the namespace that DNS has not synchronized. In this case, an application has to have a mechanism to deal with the inconsistency.

C.2.7 Requesting Information from DNS

Once an application adds its objects to the namespace and modifies the objects to contain any necessary attributes, the application is ready to use the namespace. An application can request that the DNS clerk read information stored with an object or list all the application's objects that are stored in a particular directory. An application might also need to resolve all soft links in a name in order to identify a target entry.

For example, the VAX Distributed File Service (DFS) is a layered product that provides VMS users with the ability to use remote VMS disks as if they were attached to their local VMS system. The DFS application registers VMS directory structures (a directory and all of its subdirectories) with DNS. Each DFS object registered in the namespace names a particular file-access point. DFS creates each object with a class attribute of DFS\$ACCESSPOINT and modifies the address attribute (DNS\$ADDRESS) of each object to hold the DECnet node address where the directory structures reside. As a final step in registering its resources, DFS creates a database to map DNS names to the appropriate VMS directory structures.

Whenever the DFS application receives the following mount request, DFS sends a request for information to the DNS clerk:

`MOUNT ACCESS_POINT dns-name vms-logical-name`

To read the address attribute of the access point object, the DFS application performs the following procedures:

1. Translates the DNS name that is supplied through the user interface to opaque format using the \$DNS parse function
2. Reads the class attribute of the object with the \$DNS read attribute function, indicating that there will be a second call to read other attributes of the object

3. Makes a second call to the \$DNS service to read the address attribute of the object
4. Sends the DNS name to the DFS server, which looks up the disk where the access point is located
5. Verifies that the DNS name is valid on the DFS server

Then, the DFS client and DFS server communicate to complete the mount function.

C.2.7.1 Reading Objects

When requesting information from DNS, an application always takes an object name from the user interface, translates the name into opaque format, and passes it in an item list to the DNS clerk.

The following C program shows how an application reads an object attribute. The \$DNSW service uses an item list to return a set of objects. Then, the application calls a run-time routine to read each value in the set.

```
#include <dnsdef.h>
#include <dnsmmsg.h>
/*
 * Parameters:
 *     opaque_objname = address of opaque full name for the object
 *                     containing the attribute to be read
 *     obj_len        = length of opaque full name of the object
 *     opaque_attname = address of the opaque simple name of the
 *                     attribute to be read
 *     attname_len    = length of opaque simple name of attribute
 */
read_attribute(opaque_objname, obj_len, opaque_attname, attname_len)
unsigned char *opaque_objname;
unsigned short obj_len;
unsigned char *opaque_attname;
unsigned short attname_len;
{
    struct $dnshb iosb;          /* Used to determine DNS status */
    char objtype = dns$k_object; /* Using object entries */

    struct $dnssitmdef readitem[6]; /* Item list for system service */
    struct dsc$dscdescriptor set_dsc, value_dsc, newset_dsc, uid_dsc;

    unsigned char attvalbuf[dns$k_maxattribute]; /* To hold the attribute */
                                                /* values returned from extraction routine. */
    unsigned char attsetbuf[dns$k_maxattribute]; /* To hold the set of */
                                                /* attribute values after the return from $DNSW. */
    unsigned char uidbuf[20]; /* Needed for context of multiple reads */

    int read_status; /* Status of read attribute routine */
    int set_status; /* Status of remove value routine */
    int xx; /* General variable used by print routine */

    unsigned short setlen; /* Contains current length of set structure */
    unsigned short val_len; /* Contains length of value extracted from set */
    unsigned short uid_len; /* Contains length of UID extracted from set */

    /* Construct an item list to read values of the attribute. */ ①
    readitem[0].dns$w_itm_code = dns$_entry;
    readitem[0].dns$w_itm_size = obj_len;
    readitem[0].dns$a_itm_address = opaque_objname;

    readitem[1].dns$w_itm_code = dns$_lookingfor;
    readitem[1].dns$w_itm_size = sizeof(char);
    readitem[1].dns$a_itm_address = &objtype;
```


VMS Version 5.3 Features

Reading Objects

```
readitem[2].dns$w_itm_code = dns$_attributename;
readitem[2].dns$a_itm_address = opaque_attname;
readitem[2].dns$w_itm_size = attname_len;

readitem[3].dns$w_itm_code = dns$_outvalset;
readitem[3].dns$a_itm_ret_length = &setlen;
readitem[3].dns$w_itm_size = dns$k_maxattribute;
readitem[3].dns$a_itm_address = attsetbuf;

*((int *)&readitem[4]) = 0;

do ②
{
    read_status = sys$dnsr(0, dns$_read_attribute, &readitem, &iosb, 0, 0);
    if(read_status == SS$_NORMAL)
    {
        read_status = iosb.dns$l_dnsb_status;
    }
    if((read_status == SS$_NORMAL) || (read_status == DNS$_MOREDATA))
    {
        do ③
        {
            set_dsc.dsc$w_length = setlen;
            set_dsc.dsc$a_pointer = &attsetbuf[0]; /* Address of set */
            value_dsc.dsc$w_length = dns$k_simplenamemax;
            value_dsc.dsc$a_pointer = attvalbuf; /* Buffer to hold */
                                                    /* attribute value */

            uid_dsc.dsc$w_length = 20;
            uid_dsc.dsc$a_pointer = uidbuf; /* Buffer to hold value's UID*/

            newset_dsc.dsc$w_length = dns$k_maxattribute;
            newset_dsc.dsc$a_pointer = &attsetbuf[0]; /* Same buffer for */
                                                    /* each call */

            set_status = dns$remove_first_set_value(&set_dsc, &value_dsc,
                                                    ④ &val_len, &uid_dsc,
                                                    &uid_len, &newset_dsc,
                                                    &setlen);

            if(set_status == SS$_NORMAL)
            { ⑤
                readitem[4].dns$w_itm_code = dns$_contextvartime;
                readitem[4].dns$w_itm_size = uid_len;
                readitem[4].dns$a_itm_address = uidbuf;

                *((int *)&readitem[5]) = 0;
            }
        }
    }
}
```



```

        printf("\tValue: "); ⑥
        for(xx = 0; xx < val_len; xx++)
            printf("%x ", attvalbuf[xx]);
        printf("\n");
    }
    else if (set_status != 0)
    {
        printf("Error %d returned when removing value from set\n",
            set_status);
        exit(set_status);
    }
    } while(set_status == SS$_NORMAL);
}
else
{
    printf("Error reading attribute = %d\n", read_status);
    exit(read_status);
}
} while(read_status == DNS$_MOREDATA);
}

```

The following list explains how the C program reads an object attribute:

- ① The item list contains five entries:
 - The opaque full name of the object with the attribute the program wants to read
 - The type of namespace entry to access
 - The opaque simple name of the attribute to read
 - The address of the buffer containing the set of values returned by the read operation
 - A zero to terminate the item list
- ② The loop repeatedly calls the \$DNSW service to read the values of the attribute because the first call might not return all the values. The loop executes until \$DNSW returns something other than DNS\$_MOREDATA.
- ③ This loop extracts all values from the set returned by \$DNSW, one value at a time. This routine sets up descriptors for buffers that are used by the DNS\$REMOVE_FIRST_SET_VALUE routine to extract values from the set. The loop executes until all values are extracted from the set or it encounters an error.
- ④ The DNS\$REMOVE_FIRST_SET_VALUE routine extracts a value from the set.
- ⑤ This attribute name might be the context the routine uses to read additional attributes. The attribute's UID, not its value, provides the context.
- ⑥ Finally, display the value in hexadecimal format. (You could also take the attribute name and convert it to a printable format before displaying the result.)

VMS Version 5.3 Features

Listing Information

C.2.7.2 Listing Information

The list functions of \$DNS allow applications to list the objects, subdirectories, or soft links in a specific directory. Either the asterisk (*) or question mark (?) wildcard, described in Section C.2.3.3, allows an application to search the directory on the basis of its facility name.

The values DNS returns from read or enumerate functions are in different structures. For example, an enumeration of objects returns different structures than an enumeration of directories.

The following C program shows how an application can read the objects in a directory with the \$DNS system service. It demonstrates how you parse any set that the enumerate-objects function returns with a run-time routine in order to remove the first entry from the set. The example also demonstrates how the program takes each value from the set.

```
#include <dnsdef.h>
#include <dnsmg.h>
/*
 * Parameters:
 *   fname_p   : opaque full name of the directory to enumerate
 *   fname_len : length of full name of the directory
 */
struct $dnsitmdef enumitem[4];          /* Item list for enumeration */
unsigned char setbuf[100];              /* Values from enumeration */
struct $dnsb enum_iosb;                 /* DNS status information */
int synch_event;                        /* Used for synchronous AST threads */
unsigned short setlen;                  /* Length of output in setbuf */

enumerate_objects(fname_p, fname_len)
unsigned char *fname_p;
unsigned short fname_len;
{
    int enumerate_objects_ast();

    int status;                          /* General routine status */
    int enum_status;                     /* Status of enumeration routine */

    /* Set up item list */

    enumitem[0].dns$w_itm_code = dns$_directory; /* Opaque directory name */
    enumitem[0].dns$w_itm_size = fname_len;
    enumitem[0].dns$a_itm_address = fname_p;

    enumitem[1].dns$w_itm_code = dns$_outobjects; /* output buffer */
    enumitem[1].dns$a_itm_ret_length = &setlen;
    enumitem[1].dns$w_itm_size = 100;
    enumitem[1].dns$a_itm_address = setbuf;

    *((int *)&enumitem[2]) = 0; /* Zero terminate item list */

    status = lib$get_ef(&synch_event); ❶

    if(status != SS$_NORMAL)
    {
        printf("Could not get event flag to synch AST threads\n");
        exit(status);
    }

    enum_status = sys$dns(0, dns$_enumerate_objects, &enumitem,
                          ❷   &enum_iosb, enumerate_objects_ast, setbuf);
}
```



```

if(enum_status != SS$_NORMAL) ③
{
    printf("Error enumerating objects = %d\n", enum_status);
    exit(enum_status);
}
status = sys$synch(synch_event, &enum_iosb); ④
if(status != SS$_NORMAL)
{
    printf("Synchronization with AST threads failed\n");
    exit(status);
}
}

/* AST routine parameter: */
/*      outbuf : address of buffer that contains enumerated names. */
unsigned char objnamebuf[dns$k_simplenamemax]; /* Opaque object name */
enumerate_objects_ast(outbuf)
unsigned char *outbuf;
{
    struct $dnsitmdef cvtitem[3];          /* Item list for class name */
    struct $dnsb iosb;                    /* Used for name service status information */
    struct dsc$dscdescriptor set_dsc, value_dsc, newset_dsc;

    unsigned char simplebuf[dns$k_simplestrmax]; /* Object name string */

    int enum_status; /* The status of the enumeration itself */
    int status; /* Used for checking immediate status returns */
    int set_status; /* Status of remove value routine */

    unsigned short val_len; /* Length of set value */
    unsigned short sname_len; /* Length of object name */

    enum_status = enum_iosb.dns$l_dnsb_status; /* Check status */
    if((enum_status != SS$_NORMAL) && (enum_status != DNS$_MOREDATA))
    {
        printf("Error enumerating objects = %d\n", enum_status);
        sys$setef(synch_event);
        exit(enum_status);
    }
}
do
{
    /*
     * Extract object names from output buffer one
     * value at a time. Set up descriptors for the extraction.
     */
    set_dsc.dsc$w_length = setlen; /* Contains address of */
    set_dsc.dsc$a_pointer = setbuf; /* the set whose values */
                                   /* are to be extracted */

    value_dsc.dsc$w_length = dns$k_simplenamemax;
    value_dsc.dsc$a_pointer = objnamebuf; /* To contain the */
                                           /* name of an object */
                                           /* after the extraction */

    newset_dsc.dsc$w_length = 100; /* To contain a new */
    newset_dsc.dsc$a_pointer = setbuf; /* set structure after */
                                        /* the extraction. */

    /* Call RTL routine to extract the value from the set */
    set_status = dns$remove_first_set_value(&set_dsc, &value_dsc, &val_len,
                                           0, 0, &newset_dsc, &setlen);
}

```


VMS Version 5.3 Features

Listing Information

```

if(set_status == SS$_NORMAL)
{
    cvtitem[0].dns$w_itm_code = dns$_fromsimplename;
    cvtitem[0].dns$w_itm_size = val_len;
    cvtitem[0].dns$a_itm_address = objnamebuf;

    cvtitem[1].dns$w_itm_code = dns$_tostringname;
    cvtitem[1].dns$w_itm_size = dns$_simplestrmax;
    cvtitem[1].dns$a_itm_address = simplebuf;
    cvtitem[1].dns$a_itm_ret_length = &sname_len;

    *((int *)&cvtitem[2]) = 0;

    status = sys$dns(0, dns$_simple_opaque_to_string, &cvtitem,
                    &iosb, 0, 0);

    if(status == SS$_NORMAL)
        status = iosb.dns$l_dnsb_status; /* Check for errors */
    if(status != SS$_NORMAL) /* If error, terminate processing */
    {
        printf("Converting object name to string returned %d\n",
               status);
        exit(status);
    }
    else
    {
        simplebuf[sname_len] = 0; /* Null terminate for printing */
        printf("%s\n", simplebuf);
    }

    enumitem[2].dns$w_itm_code = dns$_contextvarname;
    enumitem[2].dns$w_itm_size = val_len;
    enumitem[2].dns$a_itm_address = objnamebuf;

    *((int *)&enumitem[3]) = 0;
}
else if (set_status != 0)
{
    printf("Error %d returned when removing value from set\n",
           set_status);
    exit(set_status);
}
} while(set_status == SS$_NORMAL);
if(enum_status == DNS$_MOREDATA)
{
    enum_status = sys$dns(0, dns$_enumerate_objects, &enumitem,
                        &enum_iosb, enumerate_objects_ast, setbuf);

    if(enum_status != SS$_NORMAL) /* Check status of $DNS */
    {
        printf("Error enumerating objects = %d\n", enum_status);
        sys$setef(synch_event);
    }
}
else
{
    sys$setef(synch_event);
}
}

```

The following list explains how the C program reads objects in a directory:

- ① Get an event flag to synchronize the execution of AST threads.
- ② Use the system service to enumerate the object names.
- ③ Check the status of the system service before waiting for threads.

- ④ Use the \$SYNCH call to make sure the DNS clerk has completed and that all threads have finished executing.
- ⑤ After enumerating objects, \$DNS calls an AST routine. The routine shows how DNS\$REMOVE_FIRST_SET_VALUE extracts object names from the set returned by the DNS\$_ENUMERATE_OBJECTS function.
- ⑥ Use an item list to convert the opaque simple name to a string name so you can display it to the user. The item list contains the following entries:
 - The address of the opaque simple name to be converted
 - The address of the buffer that will hold the string name
 - A zero to terminate the item list
- ⑦ This object name could provide the context for continuing the enumeration. Append the context variable to the item list so the enumeration can continue from this name if there is more data.
- ⑧ Use the system service to enumerate the object names as long as there is more data.
- ⑨ Set the event flag to indicate that all AST threads have completed and the program can terminate.

C.2.7.3 How the Clerk Locates Data

When the DNS clerk receives an application's call for service, it tries to find a DNS server that can process the request.

Often, the DNS clerk does not know which DNS server holds the object information. To find an unknown server, the clerk looks in its own cache first. The clerk cache holds namespace information gathered from servicing earlier application requests. If the clerk cache does not list the needed server, then the DNS clerk requests information from a local DNS server in its cache. (A clerk always knows about at least one DNS server because this information is loaded at system startup.)

The clerk's last recourse is to trace directory pointers through the namespace. Any DNS server is capable of telling the clerk about another DNS server holding other directories in the namespace hierarchy. The clerk follows directory pointers until it finds a DNS server holding the specified directory. If the clerk cannot find the specified directory, then it follows directory pointers up to the root directory. Once the root directory is found, the clerk traces directory pointers away from the root, until it finds a DNS server that has the directory holding the requested object.

Once the clerk finds a directory that holds the required information, it delivers the request to the DNS server. As soon as the clerk receives a response, it transmits the result to the application.

C.2.8 DNS System Services

The Distributed Name Service Clerk system services are the programming interface to the Distributed Name Service facility. The DNS Clerk system services allow an application to register a resource in a distributed database and then access the resource from any point in the network by a single name. There are two system service calls to the clerk that are described in this section.

- \$DNS (Distributed Name Service Clerk)
- \$DNSW (Distributed Name Service Clerk and Wait)

VMS Version 5.3 Features

DNS System Services

The \$DNS system service is the asynchronous client interface for applications using the Distributed Name Service. The \$DNSW system service is the synchronous client interface.

\$DNS Distributed Name Service Clerk

The Distributed Name Service Clerk service registers a resource in a distributed database. The \$DNS service completes asynchronously; that is, it returns to the client immediately after making a name service call. The status returned to the client call indicates whether a request was successfully queued to the name service.

Note that the Distributed Name Service Clerk and Wait (\$DNSW) call is the synchronous equivalent of \$DNS. \$DNSW is identical to \$DNS in every way except that \$DNSW returns to the caller after the operation completes.

Format

SYS\$DNS [efn] ,func ,itmlst ,[dnsb] ,[astadr] ,[astprm]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Longword condition value. All system services return by immediate value a condition value in R0. Condition values returned by this call are listed in the section Condition Values Returned. Errors returned here are from the DNS clerk. Refer to the **dnsb** argument for errors returned by the name service.

Arguments

efn
VMS Usage: ef_number
type: longword (unsigned)
access: read only
mechanism: by value

Number of the event flag to be set when \$DNS completes. The **efn** argument is a longword containing this number. The **efn** argument is optional; if not specified, event flag 0 is set.

When \$DNS begins execution, it clears the event flag. Even if the service encounters an error and completes without queuing a name service request, the specified event flag is set.

func
VMS Usage: function_code
type: longword (unsigned)
access: read only
mechanism: by value

Function code specifying the action that \$DNS is to perform. The **func** argument is a longword containing this function code.

DNS Clerk System Service Calls

\$DNS

A single call to \$DNS can specify one function code. Most function codes require or allow for additional information to be passed in the call with the **itmlst** argument.

\$DNS Function Codes

DNS\$_CREATE_OBJECT

This request creates an object in the namespace. Initially, the entry has the attributes of DNS\$UID, DNS\$UTS, DNS\$CLASS, DNS\$ACS, and DNS\$CLASSVERSION. The name service creates the DNS\$UID, DNS\$UTS, and DNS\$ACS attributes. The client application supplies the DNS\$CLASS and DNS\$CLASSVERSION attributes. You can add additional attributes using the DNS\$_MODIFY_ATTRIBUTE function.

The DNS clerk cannot guarantee that an object has been created. Another DNS\$_CREATE_OBJECT request could supersede the object created by your call. To verify an object creation, wait until the directory is skulked and then check to see if the requested object entry is present. If the value of the directory's DNS\$ALLUPTO attribute is greater than the UID of the object entry, your object entry has been successfully created.

Creating an object in the namespace requires write access to the directory in which the object will reside.

If specified, DNS\$_OUTUID holds the UID of the created object.

You must specify the following item codes:

DNS\$_CLASS (Class_Name)
DNS\$_OBJECTNAME (Opaque_Full_Name)
DNS\$_VERSION (Class_Version)

You can specify the following input item codes:

DNS\$_CONF
DNS\$_WAIT

You can specify the following output item code:

DNS\$OUTUID (UID)

\$DNS returns the following:

SS\$_NORMAL
DNS\$_ENTRYEXISTS
DNS\$_INVALID_OBJECTNAME
DNS\$_INVALID_CLASSNAME
Any condition listed in the section Condition Values Returned.

\$DNS returns the following qualifying status:

DNS\$V_DNSB_OUTLINKED

DNS\$_DELETE_OBJECT

This request removes the specified object from the namespace. The function requires delete access to the object in question.

You must specify the following input item code:

DNS\$_OBJECTNAME (Opaque_Full_Name)

You can specify the following input item codes:

DNS\$_CONF

DNS\$_WAIT

\$DNS returns the following:

SS\$_NORMAL

DNS\$_INVALID_OBJECTNAME

Any condition listed in the section Condition Values Returned.

\$DNS returns the following qualifying status:

DNS\$V_DNSB_OUTLINKED

DNS\$_ENUMERATE_ATTRIBUTES

This request returns a set of attributes in DNS\$_OUTATTRIBUTESET that is associated with the entry. The entry type is specified in the DNS\$_LOOKINGFOR entry.

To manipulate the values returned by this call, use the DNS\$REMOVE_FIRST_SET_VALUE run-time routine. The values returned are the Enum_Att_Name structure, which is described in Table C-1.

You must have read access to the entry to enumerate its attributes.

The DNS clerk enumerates attributes in alphabetical order. A return status of DNS\$_MOREDATA implies that not all attributes have been enumerated. You should make further calls, setting DNS\$_CONTEXTVARNAME to the last attribute in the set returned, until the procedure returns SS\$_NORMAL.

You must specify the following input item codes:

DNS\$_ENTRY (Opaque_Full_Name)

DNS\$_LOOKINGFOR (Entry_Type)

You must specify the following output item code:

DNS\$_OUTATTRIBUTESET (set of Enum_Att_Name)

You can specify any of the following input item codes:

DNS\$_CONF

DNS\$_CONTEXTVARNAME (Opaque_Simple_Name)

DNS\$_WAIT

\$DNS can return the following:

SS\$_NORMAL

DNS\$_MOREDATA

DNS\$_INVALID_ENTRYNAME

DNS\$_INVALID_CONTEXTNAME

Any condition listed in the section Condition Values Returned.

\$DNS returns the following qualifying status:

DNS\$V_DNSB_OUTLINKED

DNS\$_ENUMERATE_CHILDREN

This request takes as input a directory name with an optional simple name that uses a wildcard. The DNS clerk matches the input against child directory entries in the specified directory.

The DNS clerk returns a set of simple names of child directories in the target directory that match the name with the wildcard. A null set is returned when there is no match or when the directory has no children.

DNS Clerk System Service Calls

\$DNS

To manipulate the values returned by this call, use the `DNS$REMOVE_FIRST_SET_VALUE` run-time routine. The value returned is a simple name.

The function requires read access to the parent directory.

The child directories are enumerated in alphabetical order. If the call returns `DNS$_MOREDATA`, not all children have been enumerated and the client should make further calls, setting `DNS$_CONTEXTVARNAME` to the last child directory in the set returned, until the procedure returns `SS$_NORMAL`. Subsequent calls return the child directories, starting with the directory specified in `DNS$_CONTEXTVARNAME` and continuing in alphabetical order.

You must specify the following input item code:

`DNS$_DIRECTORY` (Opaque_Full_Name)

You must specify the following output item code:

`DNS$_OUTCHILDREN` (set of Opaque_Simple_Name)

You can specify the following input item codes:

`DNS$_CONF`

`DNS$_CONTEXTVARNAME` (Opaque_Simple_Name)

`DNS$_WAIT`

`DNS$_WILDCARD` (Opaque_Simple_Name)

\$DNS returns the following:

`SS$_NORMAL`

`DNS$_MOREDATA`

`DNS$_INVALID_DIRECTORYNAME`

`DNS$_INVALID_CONTEXTNAME`

`DNS$_INVALID_WILDCARDNAME`

You might receive the following qualifying status:

`DNS$V_DNSB_OUTLINKED`

DNS\$_ENUMERATE_OBJECTS

This request takes as input the directory name, a simple name that uses a wildcard, and a class name that uses a wildcard. The DNS clerk matches these against objects in the directory. If a wildcard and class filter are not specified, then all objects in the directory are returned.

The function returns (in `DNS$_OUTOBJECTS`) a set of simple names of objects in the directory that match the name with the wildcard. If no objects match the wildcard or the directory contains no objects, a null set is returned. The DNS clerk returns `DNS$V_DNSB_OUTLINKED` qualifying status if it encounters one or more soft links in resolving the names of object entries to be enumerated.

To manipulate the values returned by this call, use the `DNS$REMOVE_FIRST_SET_VALUE` run-time routine. The value returned is a simple name structure.

This function requires read access to the parent directory.

The objects are enumerated in alphabetical order. If the call returns `DNS$_MOREDATA`, not all objects have been enumerated and the client should make further calls, setting `DNS$_CONTEXTVARNAME` to the last object in the set returned, until the procedure returns `SS$_NORMAL`. If the class filter is specified, only those objects of the specified classes are returned.

You must specify the following input item code:

DNS\$_DIRECTORY (Opaque_Full_Name)

You must specify the following output item code:

DNS\$_OUTOBJECTS (set of Opaque_Simple_Names)

You can specify any of the following input item codes:

DNS\$_WILDCARD (Opaque_Simple_Name)

DNS\$_CLASSFILTER (Opaque_Simple_Name)

DNS\$_CONTEXTVARNAME (Opaque_Simple_Name)

DNS\$_CONF

DNS\$_WAIT

\$DNS returns the following:

SS\$_NORMAL

DNS\$_MOREDATA

DNS\$_INVALID_DIRECTORYNAME

DNS\$_INVALID_CONTEXTNAME

DNS\$_INVALID_WILDCARDNAME

DNS\$_INVALID_CLASSNAME

You might receive the following qualifying status:

DNS\$V_DNSB_OUTLINKED

DNS\$_ENUMERATE_SOFTLINKS

This request takes as input the name of a directory and a simple name that includes a wildcard. The DNS clerk matches these against soft links in the directory. It returns (in DNS\$_OUTSOFTLINKS) a set consisting of simple names of soft links in the directory that match the specified simple name. If no soft link entries match the simple name that contains the wildcard or the directory contains no soft links, a null set is returned.

If no wildcard is specified, then all soft links in the directory are returned.

To manipulate the values returned by this call, use the DNS\$REMOVE_FIRST_SET_VALUE run-time routine. The value returned is a simple name.

This function requires read access to the parent directory.

The soft links are enumerated in alphabetical order. If the call returns DNS\$_MOREDATA, not all matching soft links have been enumerated and the client should make further calls, setting DNS\$_CONTEXTVARNAME to the last soft link in the set returned, until the procedure returns SS\$_NORMAL.

You must specify the following input item code:

DNS\$_DIRECTORY (Opaque_Full_Name)

You must specify the following output item code:

DNS\$_OUTSOFTLINKS (set of Opaque_Simple_Name)

You can specify the following input item codes:

DNS\$_WILDCARD (Opaque_Simple_Name)

DNS\$_CONTEXTVARNAME (Opaque_Simple_Name)

DNS\$_CONF

DNS\$_WAIT

DNS Clerk System Service Calls

\$DNS

\$DNS returns the following:

SS\$ _NORMAL
DNS\$ _INVALID_DIRECTORYNAME
DNS\$ _INVALID_CONTEXTNAME
DNS\$ _INVALID_WILDCARDNAME

You might receive the following qualifying status:

DNS\$V_DNSB_OUTLINKED

DNS\$ _FULL_OPAQUE_TO_STRING

This request converts a full name in opaque format to its equivalent in string format, as described in Section C.2.2.4. Setting the byte referred to by DNS\$ _SUPPRESS_NSNAME to 1 prevents the namespace name from being included in the string name.

You must specify the following item codes:

DNS\$ _FROMFULLNAME (Opaque_Full_Name)
DNS\$ _TOSTRINGNAME (Full_Name_Str)

You can specify the following input item code:

DNS\$ _SUPPRESS_NSNAME (byte)

\$DNS returns the following:

SS\$ _NORMAL
DNS\$ _INVALIDNAME

You do not receive qualifying status.

DNS\$ _MODIFY_ATTRIBUTE

This request applies one update to the specified entry in the namespace. You can add or remove an attribute; you can add or remove a value from either a single-value attribute or a set-valued attribute.

This operation requires write or delete access to the entry whose attribute is being modified, depending on whether the operation adds or removes the attribute.

When adding a value to a single-value attribute, include a value in DNS\$ _MODVALUE or you will receive the error DNS\$ _INVALIDUPDATE. The item code DNS\$ _MODVALUE is not required when writing to an attribute set because the name service creates the attribute if no value is provided.

In a delete operation, include the DNS\$ _MODVALUE item code to remove a certain value from an attribute set. Unless you specify the item code, the name service removes the attribute and all its values from the entry.

You must specify the following item codes:

DNS\$ _ENTRY (Opaque_Full_Name)
DNS\$ _LOOKINGFOR (Entry_Type)
DNS\$ _MODOPERATION (DNS\$K_PRESENT or DNS\$K_ABSENT)
DNS\$ _ATTRIBUTETYPE (DNS\$K_SET or DNS\$K_SINGLE)
DNS\$ _ATTRIBUTENAME (Opaque_Simple_Name)

You can specify the following input item codes:

DNS\$ _CONF
DNS\$ _MODVALUE

DNS\$_WAIT

\$DNS returns the following:

SS\$_NORMAL
DNS\$_WRONGATTRIBUTETYPE
DNS\$_INVALIDUPDATE
DNS\$_INVALID_ENTRYNAME
DNS\$_INVALID_ATTRIBUTENAME

You might receive the following qualifying status:

DNS\$_V_DNSB_OUTLINKED

DNS\$_PARSE_FULLNAME_STRING

This request takes a full name in string format and converts it to its equivalent in opaque format. If DNS\$_NEXTCHAR_PTR is used, the longword referenced by this entry contains the address of the character immediately following the DNS name given in DNS\$_FROMSTRINGNAME.

You must specify the following item codes:

DNS\$_FROMSTRINGNAME (Full_Name_Str)
DNS\$_TOFULLNAME (Opaque_Full_Name)

You can specify the following input item code:

DNS\$_NEXTCHAR_PTR

\$DNS can return the following:

SS\$_NORMAL
DNS\$_INVALIDNAME

You do not receive qualifying status.

DNS\$_PARSE_SIMPLENAME_STRING

This request takes a simple name in string format and converts it to its equivalent in opaque format. If DNS\$_NEXTCHAR_PTR is used, the longword referenced by this entry contains the address of the character immediately following the DNS name given in DNS\$_FROMSTRINGNAME.

You must specify the following item codes:

DNS\$_FROMSTRINGNAME (Simple_Name_Str)
DNS\$_TOFULLNAME (Opaque_Simple_Name)

You can specify the following input item code:

DNS\$_NEXTCHAR_PTR

\$DNS can return the following:

SS\$_NORMAL
DNS\$_INVALIDNAME

You do not receive qualifying status.

DNS\$_READ_ATTRIBUTE

This request returns (in DNS\$_OUTVALSET) a set whose members are the values of the specified attribute. When the request completes successfully, the qualifying status indicates whether soft links were followed in resolving the name.

DNS Clerk System Service Calls

\$DNS

This function requires read access to the object whose attribute is to be read.

To manipulate the values returned by this call, use the `DNS$REMOVE_FIRST_SET_VALUE` run-time routine. The contents of `DNS$_OUTVALSET` are passed to `DNS$REMOVE_FIRST_SET_VALUE`, and the routine returns the value of the attribute.

The attribute values are returned in the order they were received. If the call returns `DNS$_MOREDATA`, not all values have been returned. The client application can make further calls, setting `DNS$_CONTEXTVARTIME` to the time-stamping of the last attribute in the set returned, until the procedure returns `SS$_NORMAL`. If the client sets the `DNS$_MAYBEMORE` argument to 1, the name service attempts to make subsequent `DNS$_READ_ATTRIBUTE` calls for the same entry more efficient. The client may set this argument to true on any call, but performance improves only if the client accesses no other entry before making a read attribute call for the previous entry.

You must include the following input item codes:

- `DNS$_ENTRY` (Opaque_Full_Name)
- `DNS$_LOOKINGFOR` (Entry_Type)
- `DNS$_ATTRIBUTENAME` (Opaque_Simple_Name)

You must include the following output item code:

- `DNS$_OUTVALSET` (set of values)

You can include the following input item codes:

- `DNS$_MAYBEMORE` (Boolean)
- `DNS$_CONTEXTVARTIME` (UID)
- `DNS$_CONF`
- `DNS$_WAIT`

`$DNS` returns the following:

- `SS$_NORMAL`
- `DNS$_MOREDATA`
- `DNS$_INVALID_ENTRYNAME`
- `DNS$_INVALID_ATTRIBUTENAME`

You might receive the following qualifying status:

- `DNS$V_DNSB_OUTLINKED`

DNS\$_RESOLVE_NAME

This request follows a chain of soft links to its destination, returning the full name of that entry so that future calls by the client application can use the entry name without incurring the overhead of following the link.

This function requires read access to each of the soft links in the chain.

Applications that maintain their own databases of opaque DNS names should use `DNS$_RESOLVE_NAME` any time they receive the qualifying status `DNS$V_DNSB_OUTLINKED`. This status indicates a need to update the current name, using the soft link facility of DNS. Use the original name with `DNS$_RESOLVE_NAME` and store the result in the application database.

If the application provides a name that does not contain any soft links, `DNS$_NOTLINKED` status is returned. If the target of any of the chain of soft links followed does not exist, the `DNS$_DANGLINGLINK` status is returned. To obtain the target of any particular soft link, use the `DNS$_READ_ATTRIBUTE` function

with `DNS$_LOOKINGFOR` set to `DNS$_K_SOFTLINK` and request the attribute `DNS$_LINKTARGET`. This can be useful in discovering which link in a chain is "broken." If the DNS clerk detects a loop, it returns `DNS$_POSSIBLECYCLE` status.

You must specify the following input item code:

`DNS$_LINKNAME` (Opaque_Full_Name)

You must specify the following output item code:

`DNS$_OUTNAME` (Opaque_Full_Name)

You can specify the following input item codes:

`DNS$_CONF`

`DNS$_WAIT`

`$DNS` returns the following:

`SS$_NORMAL`

`DNS$_INVALID_LINKNAME`

`DNS$_NOTLINKED`

You might receive the following qualifying status:

`DNS$V_DNSB_OUTLINKED`

`DNS$_SIMPLE_OPAQUE_TO_STRING`

This request takes a simple name in opaque format and converts it to its equivalent in string format, as described in Section C.2.2.4.

You must specify the following item codes:

`DNS$_FROMSIMPLENAME` (Opaque_Simple_Name)

`DNS$_TOSTRINGNAME` (Simple_Name_Str)

`$DNS` returns the following:

`SS$_NORMAL`

`DNS$_INVALIDNAME`

You do not receive qualifying status.

`DNS$_TEST_ATTRIBUTE`

This request returns `DNS$_TRUE` if the specified attribute has one of the following characteristics:

- It is a single-value attribute and its value matches the client-specified value.
- It is a set-valued attribute and the attribute contains the client-specified value as one of its members.

On successful completion of the function, `DNS$V_DNSB_OUTLINKED` indicates whether soft links were followed in resolving the name.

This function requires test or read access to the entry whose attribute is to be tested.

If the attribute is not present in the entry or if the requested attribute does not exist, the function returns `DNS$_FALSE`.

You must specify the following item codes:

`DNS$_ENTRY` (Opaque_Full_Name)

`DNS$_LOOKINGFOR` (Entry_Type)

DNS Clerk System Service Calls

\$DNS

DNS\$_ATTRIBUTENAME (Opaque_Simple_Name)
DNS\$_VALUE (value)

You can specify the following input item codes:

DNS\$_CONF
DNS\$_WAIT

\$DNS returns the following when the call is successful:

DNS\$_TRUE
DNS\$_FALSE

\$DNS returns the following when the call is unsuccessful:

DNS\$_INVALID_ENTRYNAME
DNS\$_INVALID_ATTRIBUTENAME

You might receive the following qualifying status:

DNS\$V_DNSB_OUTLINKED

DNS\$_TEST_GROUP

This request tests for group membership. It returns DNS\$_TRUE if the specified member is a member of the specified group (or a subgroup thereof) and DNS\$_FALSE otherwise. If a recursive search is required and one or more of the subgroups is unavailable, the status encountered in trying to access that group is returned.

The DNS\$_INOUTDIRECT argument, on input, controls the scope of the search. If set to true, the only group considered is the top-level group specified by the group argument. If set to false, recursive evaluation is performed. On output, the DNS\$_INOUTDIRECT argument is set to 1 if the member was found in the top level group; otherwise, it is set to 0.

You must specify the following item codes:

DNS\$_GROUP (Opaque_Full_Name)
DNS\$_MEMBER (Opaque_Full_Name)

You can specify the following input item codes:

DNS\$_CONF
DNS\$_INOUTDIRECT (Boolean)
DNS\$_WAIT

\$DNS returns the following:

SS\$_NORMAL
DNS\$_NOTAGROUP
DNS\$_INVALID_GROUPNAME
DNS\$_INVALID_MEMBERNAME

You might receive the following qualifying status:

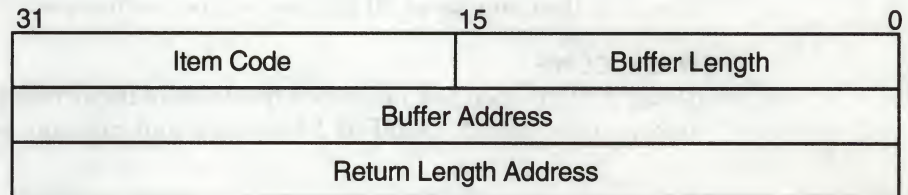
DNS\$V_DNSB_INOUTDIRECT

itmlst

VMS Usage: item_list_3
type: longword (unsigned)
access: read only
mechanism: by reference

Item list supplying information to be used in performing the function specified by the **func** argument. The **itmlst** argument is the address of the item list. The item list consists of one or more item descriptors, each of which is three longwords. The descriptors can be in any order in the item list. Each item descriptor specifies an item code. Each item code either describes the specific information to be returned by \$DNS or otherwise affects the action designated by the function code. The item list is terminated by a longword of zero.

The item list is in standard VMS format. The following figure depicts the general structure of an item descriptor:



ZK-1705-GE

\$DNS Item Descriptor Fields

item code

A word containing a symbolic code describing the nature of the information currently in the buffer or to be returned in the buffer. The location of the buffer is pointed to by the buffer address field. Each item code has a symbolic name; these symbolic names are defined by the \$DNS macro and have the format DNS\$_code.

buffer length

A word specifying the length of the buffer; the buffer either supplies information to be used by \$DNS or receives information from \$DNS. The required length of the buffer varies depending on the item code specified; each item code description specifies the required length.

buffer address

A longword containing the address of the buffer that specifies or receives the information.

return length address

A longword containing the address of a word specifying the actual length in bytes of the information returned by \$DNS. The information resides in a buffer identified by the buffer address field. The field applies to output item-list entries only and must be zero for input entries. If the return-length address is 0, it is ignored.

\$DNS Item Codes

DNS\$_ATTRIBUTETYPE

The DNS\$_ATTRIBUTETYPE item code specifies whether an attribute is set valued (DNS\$_K_SET) with a value of 3 or single valued (DNS\$_K_SINGLE) with a value of 2.

DNS\$_ATTRIBUTENAME

The DNS\$_ATTRIBUTENAME item code specifies the opaque simple name of an attribute. An attribute name cannot be longer than 31 characters.

DNS Clerk System Service Calls

\$DNS

DNS\$_CLASS

The DNS\$_CLASS item code specifies the class of an object for the \$DNS function DNS\$_CREATE_OBJECT. DNS\$_CLASS is an opaque simple name.

DNS\$_CLASSFILTER

DNS\$_CLASSFILTER is used by the \$DNS function DNS\$_ENUMERATE_OBJECTS to limit the scope of the enumeration to those objects belonging to a certain class (or, if a wildcard name is used, a group of classes). DNS\$_CLASSFILTER is an opaque simple name, which can use a wildcard.

DNS\$_CLASSFILTER is optional. A wildcard simple name of * is used by default, meaning that objects of all classes will be enumerated.

DNS\$_CONF

DNS\$_CONF specifies for \$DNS the level of importance in returning up-to-date information. DNS\$_CONF is 1 byte long and can take one of the following values:

Confidence Level	Value	Meaning
DNS\$K_LOW	1	Service the DNS clerk request at the lowest cost, usually from cached information.
DNS\$K_MEDIUM	2	Bypass any cached information and obtain the data directly from a name server.
DNS\$K_HIGH	3	Service the request from a master directory.

The entry is optional; if it is not specified, the DNS clerk assumes a value of DNS\$K_LOW.

DNS\$_CONTEXTVARNAME

DNS\$_CONTEXTVARNAME is used by the enumeration functions of \$DNS to specify a context from which the enumeration is to begin. The item is an opaque simple name.

DNS\$_CONTEXTVARNAME is optional. If not given, the enumeration begins with the first element.

DNS\$_DIRECTORY

DNS\$_DIRECTORY is used by most of the enumeration functions of \$DNS to specify the namespace directory in which the elements of the enumeration are to be found. DNS\$_DIRECTORY is an opaque full name.

DNS\$_ENTRY

DNS\$_ENTRY specifies for \$DNS the opaque full name of a namespace entry (object, soft link, directory, clearinghouse).

DNS\$_FROMFULLNAME

DNS\$_FROMFULLNAME specifies for the DNS\$_FULL_OPAQUE_TO_STRING function the opaque full name that is to be converted into string format.

DNS\$_FROMSIMPLENAME

DNS\$_FROMSIMPLENAME specifies for the DNS\$_SIMPLE_OPAQUE_TO_STRING function the opaque simple name that is to be converted into string format.

DNS\$_FROMSTRINGNAME

DNS\$_FROMSTRINGNAME specifies a name in string format for the parse functions DNS\$_PARSE_FULLNAME_STRING and DNS\$_PARSE_SIMPLENAME_STRING that is to be converted to opaque format.

DNS\$_GROUP

DNS\$_GROUP specifies for the DNS\$_TEST_GROUP function the opaque full name of the group that is to be tested. DNS\$_GROUP must be the name of a group object.

DNS\$_INOUTDIRECT

DNS\$_INOUTDIRECT is a Boolean value that serves two different purposes for the DNS\$_TEST_GROUP function. On input, DNS\$_INOUTDIRECT controls the scope of the search for the test, as follows:

Value	Definition
1	The only group to be tested is the top-level group specified by the DNS\$_GROUP item.
0	All subgroups of the group named in DNS\$_GROUP are tested for inclusion. A subgroup is any member that is a group in itself.

On output, DNS\$_INOUTDIRECT is set to indicate whether the members were found in the top-level group or were found as members of one of the subgroups, as follows:

Value	Definition
1	The member was found in the top-level group.
0	The member was found in one of the subgroups of the top-level group.

DNS\$_INOUTDIRECT is a single-byte value.

DNS\$_LINKNAME

DNS\$_LINKNAME specifies the opaque full name of a soft link.

DNS\$_LOOKINGFOR

DNS\$_LOOKINGFOR specifies the type of entry on which the call is to operate. DNS\$_LOOKINGFOR, which is encoded as a byte, can take one of the following values:

Type of Entry	Value
DNS\$_K_DIRECTORY	1
DNS\$_K_OBJECT	2
DNS\$_K_CHILDDIRECTORY	3
DNS\$_K_SOFTLINK	4
DNS\$_K_CLEARINGHOUSE	5

DNS\$_MAYBEMORE

DNS\$_MAYBEMORE is used with the DNS\$_READ_ATTRIBUTE function to indicate that the results of the read operation are to be cached. This is a single-byte item.

DNS Clerk System Service Calls

\$DNS

When this item is set to 1, the name service returns as much information about the attributes for the entry as it is able to fit in the return buffer. All of this information is cached to make later lookups of attribute information for the entry quicker and more efficient.

If this item is not supplied, then only the requested information for the entry is returned.

DNS\$_MEMBER

DNS\$_MEMBER specifies for the DNS\$_TEST_GROUP function of \$DNS the opaque full name of a member that is to be tested for inclusion within a given group.

DNS\$_MODOPERATION

DNS\$_MODOPERATION specifies for the DNS\$_MODIFY_ATTRIBUTE function the type of operation that is to take place. There are two types of modifications: adding an attribute (DNS\$K_PRESENT), which has a value of 1, or deleting an attribute (DNS\$K_ABSENT), which has a value of 0.

The name service adds an attribute in the following way:

- For an existing attribute where an attribute value is given, the value is added to a set-valued attribute and all other values for the set are unaffected. The value replaces any previous value in a single-value attribute.
- For an existing attribute where an attribute value is not given, all previous values for the attribute are unaffected.
- For a new attribute
 - Where an attribute is given, the attribute is created and given the attribute type of DNS\$K_SET or DNS\$K_SINGLE as supplied with the DNS\$K_ATTRIBUTE_TYPE item. The value is assigned to the attribute.
 - Where an attribute value is not given, a set-valued attribute is created without a value assignment, but a single-value attribute is *not* created.

The name service deletes an attribute in the following way:

- If the attribute exists and an attribute value is given, the attribute value is removed from a set-valued attribute. All other values are unaffected. For a single-value attribute, the attribute (along with its value) is removed from the entry.
- If an attribute value is not given, then the attribute and all values of the attribute are removed. This is true for both set-valued attributes and single-value attributes.

DNS\$_MODVALUE

DNS\$_MODVALUE specifies for the DNS\$_MODIFY_ATTRIBUTE function the value that is to be added to or deleted from an attribute. The structure of this value is dependent on the application.

DNS\$_MODVALUE is an optional argument that affects the overall operation of the DNS\$_MODIFY_ATTRIBUTE function. (See the DNS\$_MODOPERATION item code description for more information.)

DNS\$_NEXTCHAR_PTR

DNS\$_NEXTCHAR_PTR is an optional item code that can be used with the parse functions DNS\$_PARSE_FULLNAME_STRING and DNS\$_PARSE_SIMPLENAME_STRING to return the address of the character that immediately follows a valid DNS name. This option is most useful when applications are parsing command line strings.

Without this item code, the parse functions return an error if any portion of the name string is invalid.

DNS\$_OBJECTNAME

DNS\$_OBJECTNAME specifies the opaque full name of an object.

DNS\$_OUTATTRIBUTESET

DNS\$_OUTATTRIBUTESET specifies to the DNS\$_ENUMERATE_ATTRIBUTES function the address of a buffer that is to contain the set of enumerated attribute names.

The names returned in this set can be extracted from the buffer with the DNS\$REMOVE_FIRST_SET_VALUE routine. The resulting values are contained in the \$DNSATTRSPECDEF structure, a byte indicating whether an attribute is set valued or single valued and followed by an opaque simple name.

DNS\$_OUTNAME

DNS\$_OUTNAME specifies for the DNS\$_RESOLVE_NAME function the address of a buffer that is to contain the opaque full name of the namespace entry that is pointed to by a soft link.

DNS\$_OUTOBJECTS

DNS\$_OUTOBJECTS specifies for the DNS\$_ENUMERATE_OBJECTS function the address of a buffer that is to contain the set of opaque simple names returned by the enumeration.

The values resulting from the enumeration can be extracted using the DNS\$REMOVE_FIRST_SET_VALUE routine. The resulting values are the opaque simple names of the objects found in the directory.

DNS\$_OUTCHILDREN

DNS\$_OUTCHILDREN specifies for the DNS\$_ENUMERATE_CHILDREN function the address of a buffer that is to contain the set of opaque simple names returned by the enumeration.

The values resulting from the enumeration can be extracted using the DNS\$REMOVE_FIRST_SET_VALUE routine. These values are the opaque simple names of the child directories found in the parent directory.

DNS\$_OUTSOFTLINKS

DNS\$_OUTSOFTLINKS specifies for the DNS\$_ENUMERATE_SOFTLINKS function the address of a buffer that is to contain the set of opaque simple names returned by the enumeration.

The values resulting from the enumeration can be extracted using the DNS\$REMOVE_FIRST_SET_VALUE routine. The resulting values are the opaque simple names of the soft links found in the directory.

DNS\$_OUTVALSET

DNS\$_OUTVALSET specifies for the DNS\$_READ_ATTRIBUTE function the address of a buffer that is to contain the set of values for the given attribute.

DNS Clerk System Service Calls

\$DNS

The values of the set placed in this buffer can be extracted using the `DNS$REMOVE_FIRST_SET_VALUE` routine. The extracted values are the values of the attribute.

DNS\$_OUTUID

`DNS$_OUTUID` is an optional item code that contains the address of a buffer used by the create functions of `$DNS` to return the unique identifier (UID). The UID is the time-stamping the entry received at creation.

DNS\$_SUPPRESS_NSNAME

`DNS$_SUPPRESS_NSNAME` is an optional item for the `DNS$_FULL_OPAQUE_TO_STRING` function that is used to indicate that the leading namespace name should not be returned in the converted full name string. This is a single-byte value.

A value of 1 suppresses the leading namespace name in the resulting full name string.

DNS\$_TOFULLNAME

`DNS$_TOFULLNAME` specifies for the `DNS$_PARSE_FULLNAME_STRING` function the address of a buffer that will contain the resulting opaque full name.

DNS\$_TOSIMPLENAME

`DNS$_TOSIMPLENAME` specifies for the `DNS$_PARSE_SIMPLENAME_STRING` function the address of a buffer that will contain the resulting opaque simple name.

DNS\$_TOSTRINGNAME

`DNS$_TOSTRINGNAME` specifies the address of a buffer that is to contain the string name resulting from one of the conversion functions: `DNS$_FULL_OPAQUE_TO_STRING` or `DNS$_SIMPLE_OPAQUE_TO_STRING`.

DNS\$_VALUE

`DNS$_VALUE` specifies for the `DNS$_TEST_ATTRIBUTE` function the value that is to be tested. This item contains the address of a buffer holding the value.

DNS\$_VERSION

`DNS$_VERSION` specifies for the `DNS$_CREATE_OBJECT` function the version associated with an object. This item contains the address of a `$DNCSVERSDEF` (CLASSVERSION) structure. This is a 2-byte structure: the first byte contains the major version number; the second contains the minor version number.

DNS\$_WAIT

`DNS$_WAIT` enables the client to specify a timeout value to wait for a call to complete. If the timeout expires, the call returns either `DNS$K_TIMEOUTNOTDONE` or `DNS$K_TIMEOUTMAYBEDONE`, depending on whether the namespace was updated by the incomplete operation.

The `$BINTIM` service converts an ASCII string time value to the quadword time value required by `$DNS`.

The parameter is optional; if it is not specified, a system-defined default timeout value of 10 minutes is assumed.

DNS\$_WILDCARD

DNS\$_WILDCARD is an optional item code that specifies to the enumeration functions of \$DNS the opaque simple name used to limit the scope of the enumeration. (The simple name does not have to use a wildcard.) Only those simple names that match the wildcard are returned by the enumeration.

Item Code Identifiers

The identifiers shown in Table C-1 are data structures that are used in item-code arguments. Each data structure defines the encoding of an item-list element.

Table C-1 DNS Item-Code Arguments

Item-Code Identifier	Description
Attribute_Name	The structure of an opaque simple name, limited to 31 ISO Latin 1 characters.
Attribute_Name_Str	An attribute name string with the structure of a simple name string but limited to 31 ISO Latin 1 characters.
Boolean	A 1-byte field with the value 0 if false and 1 if true.
Class_Name	An opaque simple name, limited to 31 ISO Latin 1 characters.
Class_Name_Str	A simple name string, limited to 31 ISO Latin 1 characters.
Class_Version	A 2-byte field specifying major and minor version numbers associated with the object class.
Confidence	A 1-byte field with the value: DNS\$K_LOW, DNS\$K_MEDIUM, or DNS\$K_HIGH.
Entry_Type	A 1-byte field with the value DNS\$K_OBJECT, DNS\$K_SOFTLINK, DNS\$K_DIRECTORY, or DNS\$K_CLEARINGHOUSE.
Enum_Att_Name	A structure consisting of a single byte, indicating whether the attribute is a set (DNS\$K_SET) or a single value (DNS\$K_SINGLE), followed by an opaque simple name.
Full_Name_String	A full name string with the following structure: [NS_name:] [.] namestring [.namestring]

(continued on next page)

Table C-1 (Cont.) DNS Item-Code Arguments

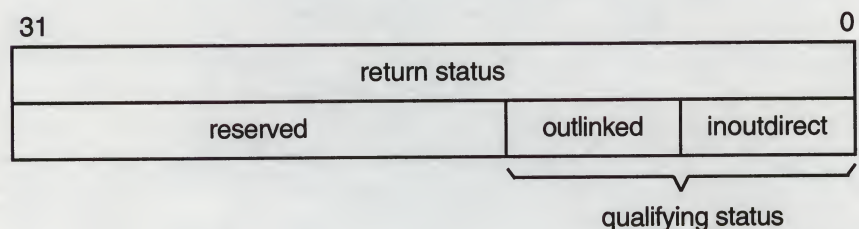
Item-Code Identifier	Description
	NS_name:, if present, is a local system representation of the NSUID, the unique identifier of the name server. The DNS clerk supplies a namespace name (<i>node-name_NS</i>) if the value is omitted.
	Namestring represents a simple name component. Multiple simple names are separated by periods. You can include the asterisk wildcard (*) and simple name strings within quotation marks.
Group_Member	A structure consisting of a single byte, indicating whether the entry is a principal (DNS\$K_GRPMEM_NOT_GROUP) or another group (DNS\$K_GRPMEM_IS_GROUP), followed by the opaque full name of the member.
Opaque_Full_Name	The internal format of the complete name identifier for an object. The maximum output of DNS\$PARSE_FULLNAME_STRING is 402 bytes.
Opaque_Simple_Name	A simple name specifies the internal format of one component of an Opaque_Full_Name. The Opaque_Simple_Name is the output of the DNS\$PARSE_SIMPLENAME_STRING routine. It can be no longer than 257 bytes.
Simple_Name_Str	One term consisting of a string of ASCII characters with its length stored separately in an item list.

dnsb

```
VMS Usage:  dns_status_block
type:       quadword (unsigned)
access:     write only
mechanism:  by reference
```

Status block to receive the final completion status of the \$DNS operation. The **dnsb** argument is the address of the quadword \$DNS status block.

The following figure depicts the structure of a \$DNS status block:



ZK-1080A-GE

Status Block Fields

return status

Set on completion of a DNS clerk request to indicate the success or failure of the operation. Check the qualifying status word for additional information about a request marked as successful. Wherever possible, each function code description includes return status values.

qualifying status

This field consists of a set of flags that provide additional information about a successful name service operation. Wherever possible, each function code description includes qualifying status values.

The qualifying status values are defined as follows:

- **DNS\$V_DNSB_INOUTDIRECT**—If true, indicates only the top-level group was searched for a member.
- **DNS\$V_DNSB_OUTLINKED**—If set, indicates that one or more soft links were encountered while resolving the object of the call.

astadr

VMS Usage: ast_procedure
type: procedure entry mask
access: call without stack unwinding
mechanism: by reference

Asynchronous system trap (AST) routine to be executed when I/O completes. The **astadr** argument, which is the address of a longword value, is the entry mask to the AST routine.

The AST routine executes in the access mode of the caller of \$DNS.

astprm

VMS Usage: user_arg
type: longword (unsigned)
access: read only
mechanism: by value

Asynchronous system trap (AST) parameter passed to the AST service routine. The **astprm** argument is a longword value containing the AST parameter.

Description

The VMS Distributed Name Service Clerk system service provides a low-level interface between an application (client) and the Distributed Name Service facility. The DNS clerk interface is used to create, delete, modify, and retrieve objects or soft links in a namespace.

A single system service call supports the DNS clerk. It has two main parameters:

- A function code identifying the particular service to perform
- An item list specifying all the parameters for the required function

The use of this item list is similar to that of other system services that use a single item list for both input and output operations.

DNS Clerk System Service Calls

\$DNS

Item-list entries must be specified in opaque format. You can convert any one of the name strings to opaque format with a conversion function. If applications need to store names, they must store them in opaque format. The opaque format guarantees the uniqueness of a name over time, whereas a string format does not.

Many of the functions return results as a set. In some cases, the specified output buffer might not be large enough to contain the complete set. In this case, the return status indicates this condition with the success status \$DNS_MOREDATA. To obtain the remaining data from the set, the client should make repeated calls, each time specifying the last attribute received in the context-variable item until the call returns SS\$_NORMAL.

The context-variable item can take one of two forms depending on the function:

- **DNS\$CONTEXTVARNAME**—If the returned data is a set of names, then the item is a simple name.
- **DNS\$CONTEXTVARTIME**—If the returned data is a set of values, then the item is a time-stamping.

If the context-variable item is not specified or is null, then the results are returned from the beginning of the set.

All functions return the SS\$_NORMAL status for success except DNS\$_TEST_ATTRIBUTE, which returns DNS\$_TRUE or DNS\$_FALSE. The functions return linked information in the \$DNS status block. The DNS\$V_DNSB_OUTLINKED bit in the status block indicates whether any soft links are encountered in an information search.

Condition Values Returned

SS\$_BADPARAM	Bad parameter value.
SS\$_NORMAL	Normal completion of the request.
DNS\$_ACCESSDENIED	Caller does not have required access to the entry in question. This error is returned only if the client has some access to the entry. Otherwise, the unknown entry status is returned.
DNS\$_BADCLOCK	The clock at the name server has a value outside the permissible range.
DNS\$_BADEPOCH	Copies of directories are not synchronized.
DNS\$_BADITEMBUFFER	Invalid output item buffer detected. (This normally indicates that the buffer has been modified during the call.)
DNS\$_CACHELOCKED	Global client cache locked.
DNS\$_CLEARINGHOUSEDOWN	Clearinghouse is not available.
DNS\$_CLERKBUG	Internal clerk error detected.

DNS\$_CONFLICTINGARGUMENTS	Two or more optional arguments conflict; they cannot be specified in the same function call.
DNS\$_DANGLINGLINK	Soft link points to nonexistent entry.
DNS\$_DATACORRUPTION	An error occurred in accessing the data stored at a clearinghouse. The clearinghouse may be corrupted.
DNS\$_ENTRYEXISTS	An entry with the same full name already exists in the namespace.
DNS\$_FALSE	Unsuccessful test operation.
DNS\$_INVALIDARGUMENT	A syntactically incorrect, out of range, or otherwise inappropriate argument was specified in the call.
DNS\$_INVALID_ATTRIBUTENAME	The name given for function is not a valid DNS attribute name.
DNS\$_INVALID_CLASSNAME	The name given for function is not a valid DNS class name.
DNS\$_INVALID_CLEARINGHOUSENAME	The name given for function is not a valid DNS clearinghouse name.
DNS\$_INVALID_CONTEXTNAME	The name given for function is not a valid DNS name.
DNS\$_INVALID_DIRECTORYNAME	The name given for function is not a valid DNS directory name.
DNS\$_INVALID_ENTRYNAME	The name given for function is not a valid DNS entry name.
DNS\$_INVALIDFUNCTION	Invalid function specified.
DNS\$_INVALID_GROUPNAME	The name given for function is not a valid DNS group name.
DNS\$_INVALIDITEM	Invalid item list entry specified.
DNS\$_INVALID_LINKNAME	The name given for function is not a valid DNS link name.
DNS\$_INVALID_MEMBERNAME	The name given for function is not a valid DNS name.
DNS\$_INVALIDNAME	A badly formed name was supplied to the call.
DNS\$_INVALID_NSNAME	Namespace name given in name string is not a valid DNS name.
DNS\$_INVALID_OBJECTNAME	The name given for function is not a valid DNS object name.
DNS\$_INVALID_TARGETNAME	The name given for function is not a valid DNS name.
DNS\$_INVALIDUPDATE	An update was attempted to an attribute that cannot be directly modified by the client.
DNS\$_INVALID_WILDCARDNAME	The name given for function is not a valid DNS name.

DNS Clerk System Service Calls

\$DNS

DNS\$_LOGICAL_ERROR	Error translating logical name in given string.
DNS\$_MISSINGITEM	Required item-list entry is missing.
DNS\$_MOREDATA	More output data to be returned.
DNS\$_NAMESERVERBUG	A name server encountered an implementation bug. Please submit an SPR.
DNS\$_NOCACHE	Client cache file not initialized.
DNS\$_NOCOMMUNICATION	No communication was possible with any name server capable of processing the request. Check NCP event 353.5 for the DECnet error.
DNS\$_NONSRESOURCES	The call could not be performed due to lack of memory or communication resources at the local node to process the request.
DNS\$_NONSNAME	Unknown namespace name specified.
DNS\$_NOTAGROUP	The full name given is not the name of a group.
DNS\$_NOTIMPLEMENTED	This function is defined by the architecture as optional and is not available in this implementation.
DNS\$_NOTLINKED	A link is not contained in the name.
DNS\$_NOTNAMESERVER	The node contacted by the clerk does not have a DNS server running. This can happen when the application supplies the clerk with inaccurate replica information.
DNS\$_NOTSUPPORTED	This version of the architecture does not support the requested function.
DNS\$_POSSIBLECYCLE	Loop detected in link or group entry.
DNS\$_RESOURCEERROR	Failure to obtain system resource.
DNS\$_TIMEOUTNOTDONE	The operation did not complete in the time allotted. No modifications have been performed even if the operation requested them.
DNS\$_TIMEOUTMAYBEDONE	The operation did not complete in the time allotted. Modifications may or may not have been made to the namespace.
DNS\$_TRUE	Successful test operation.
DNS\$_UNKNOWNCLEARINGHOUSE	The clearinghouse does not exist.
DNS\$_UNKNOWNENTRY	Either the requested entry does not exist or the client does not have access to the entry.
DNS\$_UNTRUSTEDCH	A DNS server is not included in the object's access control set.

DNS\$_WRONGATTRIBUTETYPE

The caller specified an attribute type that did not match the actual type of the attribute.

\$DNSW

Distributed Name Service Clerk and Wait

The Distributed Name Service Clerk and Wait service registers a resource in a distributed database; same as \$DNS. However, the \$DNSW service completes synchronously; that is, it returns to the caller after the operation completes.

For asynchronous completion, use the \$DNS service, which returns to the caller immediately after making a name service call. The return status to the client call indicates whether a request was successfully queued to the name service.

In all other respects, \$DNSW is identical to \$DNS. Refer to the \$DNS description for complete information about the \$DNSW service.

Format

`SYS$DNSW [efn] ,func ,itmlst ,[dnsb] ,[astadr] ,[astprm]`

C.2.9 DNS Run-Time Routines

All applications designed to take advantage of the Distributed Name Service (DNS) facility use the DNS Clerk system services and the DNS run-time routines to register a resource in the DNS namespace and to modify and find information within the namespace. This section describes the run-time routines.

DNS\$APPEND_SIMPLENAME_TO_RIGHT

Append a

Simple Name to the End of a Full Name

The Append a Simple Name to the End of a Full Name routine adds an opaque simple name to the end of an opaque full name to create a new full name.

Format

DNS\$APPEND_SIMPLENAME_TO_RIGHT

fullname ,simplename ,resulting-fullname ,resulting-length

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

fullname

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The opaque full name gaining a new simple name. The **fullname** argument is the address of a descriptor pointing to the opaque full name that is to be extended.

simplename

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The opaque simple name that is appended. The **simplename** argument is the address of a descriptor pointing to an opaque simple name that is to be appended to the full name, thus creating a new full name.

resulting-fullname

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The new full name. The **resulting-fullname** argument is the address of a descriptor that points to the buffer that receives the new full name. This buffer can be the same as the buffer referred to by the **fullname** argument; however, the descriptors must be separate.

resulting-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the new full name. The **resulting-length** argument is the address of a word that receives the length of the new full name found in **resulting-fullname**.

Description

DNS\$APPEND_SIMPLENAME_TO_RIGHT adds an opaque simple name to the end of an opaque full name to create a new full name.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
DNS\$_INVALIDNAME	The name to be converted is not a valid DNS name.
0	Error appending name.

DNS\$COMPARE_FULLNAME

Compare Full Names

The Compare Full Names routine compares two opaque full names and returns the result.

Format

DNS\$COMPARE_FULLNAME fullname1 ,fullname2

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

fullname1

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

One opaque full name. The **fullname1** argument is the address of a descriptor pointing to an opaque full name.

fullname2

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

One opaque full name. The **fullname2** argument is the address of a descriptor pointing to an opaque full name.

Description

DNS\$COMPARE_FULLNAME compares two opaque full names and returns the result. First, the procedure checks the namespace UIDs of the full names as numbers. If they are unequal, the routine returns the result. If they are equal, it compares each simple name in the full name until it finds an inequality or determines that both names are the same.

Condition Values Returned

-1	fullname1 is less than fullname2 .
0	fullname1 equals fullname2 .
1	fullname1 is greater than fullname2 .

DNS\$COMPARE_SIMPLENAME

Compare Two

Simple Names

The Compare Two Simple Names routine compares two simple names, without considering case.

Format

DNS\$COMPARE_SIMPLENAME simplename1 ,simplename2

Returns

VMS Usage: cond_value
 type: longword (unsigned)
 access: write only
 mechanism: by value

Arguments

simplename1

VMS Usage: char_string
 type: character string
 access: read only
 mechanism: by descriptor

An opaque simple name. The **simplename1** argument is the address of a descriptor pointing to the first simple name.

simplename2

VMS Usage: char_string
 type: character string
 access: read only
 mechanism: by descriptor

An opaque simple name. The **simplename2** argument is the address of a descriptor pointing to the second simple name.

Description

DNS\$COMPARE_SIMPLENAME compares two simple names, without considering case. The routine determines the relationship between two opaque simple names to see if they are equal.

Condition Values Returned

SS\$_NORMAL

Normal successful completion.

-1

simplename1 is less than **simplename2**.

0

simplename1 equals **simplename2**.

1

simplename1 is greater than **simplename2**.

DNS\$CONCATENATE_NAME

Join Two Names

The Join Two Names routine joins two opaque full names to form a new full name.

Format

DNS\$CONCATENATE_NAME

fullname1 ,fullname2 ,resulting-fullname ,resulting-length

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

fullname1

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The opaque full name to be joined. The **fullname1** argument is the address of a descriptor pointing to the opaque full name.

fullname2

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The opaque full name appended to **fullname1**. The **fullname2** argument is the address of a descriptor pointing to the full name to be appended.

resulting-fullname

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The buffer where the new full name will be written. The **resulting-fullname** argument is the address of a descriptor pointing to the buffer. This buffer can be the same as the buffer referred to by the **fullname1** argument; however, the descriptors must be separate.

resulting-length

VMS Usage: word_unsigned
 type: word (unsigned)
 access: write only
 mechanism: by reference

The length of the new full name. The **resulting-length** argument is the address of a word that receives the length of the new full name found in **resulting-fullname**.

Description

DNS\$CONCATENATE_NAME joins two opaque full names to form a new opaque full name, which is placed in the buffer named by the **resulting-fullname** argument. The new full name receives the namespace name of the first opaque full name. For example, appending the full name TEST:.POP.DIR1 (**fullname2**) to DEC:.ENG.NAC (**fullname1**) results in a full name of DEC:.ENG.NAC.POP.DIR1.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
DNS\$_INVALIDNAME	The name to be converted is not a valid DNS name.
0	Error performing concatenation.

DNS\$COUNT_SIMPLENAMES

Count the Simple Names in a Full Name

The Count the Simple Names in a Full Name routine counts the number of simple names contained in an opaque full name.

Format

DNS\$COUNT_SIMPLENAMES fullname ,count

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

fullname
VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The full name to be counted. The **fullname** argument is the address of a descriptor pointing to the full name that is to be examined for the simple names it contains.

count
VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The number of simple names found in the full name. The **count** argument is the address of a word that receives the number of simple names.

Description

DNS\$COUNT_SIMPLENAMES counts the number of simple names—but not the namespace name—found in an opaque full name. The number of simple names counted is returned in the word referenced by the **count** argument. The routine is meant to be used with DNS\$REMOVE_RIGHT_SIMPLENAME and DNS\$REMOVE_LEFT_SIMPLENAME.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
DNS\$_INVALIDNAME	The name to be converted is not a valid DNS name.

DNS\$CVT_DNSADDRESS_TO_BINARY

Convert a DNS Address to a Phase IV Binary Address

The Convert a DNS Address to a Phase IV Binary Address routine takes a DNS address and returns the DECnet Phase IV node address.

Format

DNS\$CVT_DNSADDRESS_TO_BINARY dnsaddress ,binary

Returns

VMS Usage: cond_value
 type: longword (unsigned)
 access: write only
 mechanism: by value

Arguments

dnsaddress
 VMS Usage: char_string
 type: character string
 access: read only
 mechanism: by descriptor

The DNS address. The **dnsaddress** argument is the address of a descriptor pointing to the DNS address.

binary
 VMS Usage: word_unsigned
 type: word (unsigned)
 access: write only
 mechanism: by reference

The DECnet Phase IV address found in the DNS address structure. The **binary** argument is the address of a word containing the 16-bit Phase IV address of the node.

Description

DNS\$CVT_DNSADDRESS_TO_BINARY takes a DNS address and returns the DECnet Phase IV node address. The Phase IV address is returned in a word. If no Phase IV address is found in the DNS address, then the value 0 is returned as an error.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
0	No DECnet Phase IV address found.

DNS\$CVT_DNSADDRESS_TO_NODENAME**Convert a DNS Address to a Node Name**

The Convert a DNS Address to a Node Name routine takes a DNS address and returns a DECnet Phase IV node name.

Format

DNS\$CVT_DNSADDRESS_TO_NODENAME

dnsaddress ,nodename ,resulting-length

Returns

VMS Usage: cond_value
 type: longword (unsigned)
 access: write only
 mechanism: by value

Arguments**dnsaddress**

VMS Usage: char_string
 type: character string
 access: read only
 mechanism: by descriptor

The DNS address. The **dnsaddress** argument is the address of a descriptor pointing to the DNS address.

nodename

VMS Usage: char_string
 type: character string
 access: write only
 mechanism: by descriptor

The DECnet Phase IV node name. The **nodename** argument is the address of a descriptor pointing to the Phase IV node name. The memory buffer referenced by the DSC\$A_POINTER portion of this descriptor must be large enough to contain the entire Phase IV node name string, which can be up to six bytes long.

resulting-length

VMS Usage: word_unsigned
 type: word (unsigned)
 access: write only
 mechanism: by reference

The length of the node name (in bytes) after conversion. The **resulting-length** argument is a word containing the length of the node name (in bytes) after conversion.

Description

DNS\$CVT_DNSADDRESS_TO_NODENAME takes a DNS address and returns a DECnet Phase IV node name. If no Phase IV address is found, then the value 0 is returned.

Because DNS\$CVT_DNSADDRESS_TO_NODENAME calls both \$ASSIGN and \$QIOW, it can return condition values from either of these system services. The routine also returns errors detected through NETACP.

Condition Values Returned

SS\$_NORMAL

Normal successful completion.

0

No DECnet Phase IV address found.

DNS\$CVT_NODENAME_TO_DNSADDRESS

Convert a Node Name to an Address

The Convert a Node Name to a DNS Address routine takes a DECnet Phase IV node name and returns a DNS address.

Format

DNS\$CVT_NODENAME_TO_DNSADDRESS

nodename ,dnsaddress ,resulting-length

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

nodename

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The DECnet Phase IV node name. The **nodename** argument is the address of a descriptor pointing to the node name. This routine creates a DNS address containing the node address of the given Phase IV node name.

dnsaddress

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The address of a buffer containing the DNS address. The **dnsaddress** argument is the address of a descriptor pointing to the buffer address.

resulting-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the DNS address after conversion. The **resulting-length** argument is a word containing the length of the address.

Description

DNS\$CVT_NODENAME_TO_DNSADDRESS takes a DECnet Phase IV node name and returns a DNS address. The routine creates the DNS address for a given Phase IV node name.

DNS\$CVT_NODENAME_TO_DNSADDRESS calls \$ASSIGN and \$QIOW so it can return condition values from either of these system services. The routine also returns errors detected through NETACP.

Condition Values Returned

SS\$_NORMAL

Normal successful completion.

DNS\$CVT_TO_USERNAME_STRING

Convert an Opaque User Name to a String

The Convert an Opaque User Name to a String routine converts an opaque DECnet Phase IV user name into a username string.

Format

DNS\$CVT_TO_USERNAME_STRING

fullname ,username ,resulting-length

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

fullname

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The opaque full name for the DECnet Phase IV user name. The **fullname** argument is the address of a descriptor pointing to the name.

username

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The name converted to DECnet Phase IV format (node::user). The **username** argument is the address of a descriptor pointing to a buffer containing the converted name.

resulting-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the converted user name. The **resulting-length** argument is the address of a word containing the length.

Description

DNS\$CVT_TO_USERNAME_STRING converts a DNS representation of a Phase IV user name into a Phase IV username string.

If any full name other than a DNS representation of a Phase IV user name is given, the routine returns a DNS\$_INVALIDNAME error.

Condition Values Returned

SS\$_NORMAL	Procedure successfully completed.
DNS\$_ACCESSVIOLATION	Memory or other resource access violation.
DNS\$_CACHELOCKED	Global client cache locked by another process.
DNS\$_INVALIDARGUMENT	One of the arguments was incorrect, out of range, or otherwise inappropriate.
DNS\$_INVALIDNAME	The name to be converted is not a valid DNS name.
DNS\$_NOCACHE	Client cache file not initialized.
DNS\$_RESOURCEERROR	Insufficient resources on local system to process request.

DNS\$PARSE_USERNAME_STRING

Convert a User Name from String to Opaque

The Convert a User Name from String to Opaque routine converts a DECnet Phase IV user name to an opaque full name.

Format

DNS\$PARSE_USERNAME_STRING

user-string ,phase4-name ,resulting-length [,next-character-pointer]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments

user-string

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The name string to convert. The **user-string** argument is the address of a descriptor pointing to the DECnet Phase IV username string, which is in the format *node::user*.

phase4-name

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The opaque full name resulting from the conversion. The **phase4-name** argument is the address of a descriptor pointing to the buffer that is to contain an opaque full name representing a user name on a Phase IV node.

resulting-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the opaque full name. The **resulting-length** argument is the address of a word holding the length of the name returned in **phase4-name**.

next-character-pointer

VMS Usage: address
type: address
access: write only
mechanism: by reference

The character following the DNS name extracted from **user-string**. The **next-character-pointer** argument is the address of the character following the DNS name. When you use this argument, DNS\$PARSE_USERNAME_STRING returns DNS\$_INVALIDNAME when it encounters an invalid name. In such a case, **next-character-pointer** points to the first character in the name that is invalid.

Description

DNS\$PARSE_USERNAME_STRING converts a DECnet Phase IV user name to an opaque full name that represents the user name.

The **next-character-pointer** argument affects how the routine parses the string:

- When **next-character-pointer** is zero or absent, the full name string given in **user-string** must contain valid DNS characters with DNS naming syntax. If any part of the string violates this rule, the routine returns DNS\$_INVALIDNAME and the output should not be used.
- When the **next-character-pointer** argument has a nonzero value, the parsing begins at the first character referenced by **user-string** and parsing continues until one of the following occurs:
 - An invalid DNS character is found.
 - An exception to DNS syntax rules occurs.
 - All characters have been parsed.

Then the address given by **next-character-pointer** is set to the address of the character or group of characters that is invalid. It returns DNS\$_INVALIDNAME if the colons (:) separating the node name from the user name of the Phase IV name are missing.

If any part of the node portion of the DECnet Phase IV username string is not a proper DNS name, the routine returns DNS\$_INVALIDNAME regardless of the value and whether or not the **next-character-pointer** argument is supplied.

Error conditions can result from the parse routine. You can test for error conditions in any of the following ways:

- When all parts of the name are invalid, test whether **next-character-pointer** contains the same address as **user-string**. Alternatively, test whether the resulting length is zero.
- When **user-string** contains a valid DNS name, test whether **next-character-pointer** contains the address immediately following the given buffer. Alternatively, test whether the address in **next-character-pointer** minus the address of **user-string** is equal to or larger than the size of the given buffer.

DNS\$ Run-Time Routines

DNS\$PARSE_USERNAME_STRING

- When parsing a user name that has been extracted from a command line, test whether the character given at the address of **next-character-pointer** is a valid separator for the command line; for example, a space. If you find an invalid character, then part of the DNS name is invalid.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
DNS\$_ACCESSVIOLATION	Memory or other resource access violation.
DNS\$_CACHELOCKED	Global client cache locked by another process.
DNS\$_INVALIDARGUMENT	One of the arguments was incorrect, out of range, or otherwise inappropriate.
DNS\$_INVALIDNAME	The name to be converted is not a valid DNS name.
DNS\$_NOCACHE	Client cache file not initialized.
DNS\$_RESOURCEERROR	Insufficient resources on local system to process request.
0	Error creating opaque name.

DNS\$REMOVE_FIRST_SET_VALUE

Remove a Value from a Set

The Remove a Value from a Set routine extracts the first value from a set and returns the value with its creation time-stamping UID.

Format

DNS\$REMOVE_FIRST_SET_VALUE

set [,value] [,value-length] [,uid] [,uid-length] [,newset] [,newset-length]

Returns

VMS Usage: cond_value
 type: longword (unsigned)
 access: write only
 mechanism: by value

Arguments

set

VMS Usage: char_string
 type: character string
 access: read only
 mechanism: by descriptor

The set from which the value is extracted. The **set** argument is the address of a descriptor pointing to the set.

value

VMS Usage: char_string
 type: character string
 access: write only
 mechanism: by descriptor

The value extracted from the set. The **value** argument is the address of a descriptor pointing to a buffer containing the value.

value-length

VMS Usage: word_unsigned
 type: word (unsigned)
 access: write only
 mechanism: by reference

The length of the value. The **value-length** argument is the address of a word holding the length of the value placed in **value**.

uid

VMS Usage: char_string
 type: character string
 access: write only
 mechanism: by descriptor

DNS\$ Run-Time Routines

DNS\$REMOVE_FIRST_SET_VALUE

The buffer holding the creation time-stamping UID of the extracted value. The **uid** argument is the address of a descriptor pointing to the buffer.

uid-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the UID placed in **uid**. The **uid-length** argument is the address of a word holding the length.

newset

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The opaque **set** without the first value. The **newset** argument is the address of a descriptor pointing to a buffer containing that set. The buffer can be the same as the one given in the **set** argument.

newset-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the new set copied to the **newset** buffer. The **newset-length** argument is the address of a word that receives the length.

Description

DNS\$REMOVE_FIRST_SET_VALUE extracts a value from a set and returns the value with its creation time-stamping UID. Use the routine to extract values from the sets returned by \$DNS and \$DNSW.

A set can contain any number of values that are relevant to a given call. The routine extracts values one at a time from the opaque set for use by a program. In order to extract all values from the set, you must use an execution loop.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
DNS\$_INVALIDARGUMENT	The set argument was incorrect, out of range, or otherwise inappropriate.
0	Set buffer is empty.
-1	Length of value , uid , or newset buffers too small.

DNS\$REMOVE_LEFT_SIMPLENAME

Strip the Simple Name on the Left from the Full Name

The Remove the Simple Name on the Left from the Full Name routine removes the leftmost simple name from an opaque full name. It returns both the simple name stripped and the new full name that results from the operation.

Format

DNS\$REMOVE_LEFT_SIMPLENAME

fullname [,resulting-fullname] [,resulting-fullname-length] [,resulting-simplename]
 [,resulting-simplename-length]

Returns

VMS Usage: cond_value
 type: longword (unsigned)
 access: write only
 mechanism: by value

Arguments

fullname

VMS Usage: char_string
 type: character string
 access: read only
 mechanism: by descriptor

The opaque full name to strip. The **fullname** argument is the address of a descriptor pointing to the opaque full name to strip. If the full name does not contain any simple names, the routine returns a value of 0 in **cond_value**.

resulting-fullname

VMS Usage: char_string
 type: character string
 access: write only
 mechanism: by descriptor

The opaque full name resulting from the operation. The **resulting-fullname** argument is the address of a descriptor pointing to the buffer containing the resulting opaque full name. This buffer can be the same as the buffer referred to by the **fullname** argument; however, the descriptors must be separate.

resulting-fullname-length

VMS Usage: word_unsigned
 type: word (unsigned)
 access: write only
 mechanism: by reference

The length of the full name that is returned. The **resulting-fullname-length** argument is the address of a word receiving the length of the full name returned in **resulting-fullname**.

DNS\$ Run-Time Routines

DNS\$REMOVE_LEFT_SIMPLENAME

resulting-simplename

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The simple name stripped from **fullname**. The **resulting-simplename** argument is the address of a descriptor pointing to the buffer containing the opaque simple name that was stripped.

resulting-simplename-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the simple name. The **resulting-simplename-length** argument is the address of a word that receives the length of the simple name returned in **resulting-simplename**.

Description

DNS\$REMOVE_LEFT_SIMPLENAME removes the leftmost simple name from an opaque full name. When **resulting-fullname** is nonzero, the full name resulting from the removal of the leftmost simple name is returned in that buffer. When **resulting-simplename** is nonzero, the simple name that was stripped from **fullname** is returned in that buffer. The namespace name is not stripped from the full name; only simple names are affected.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
DNS\$_INVALIDNAME	The name to be converted is not a valid DNS name.
-1	Error stripping name.
0	No simple name.

DNS\$REMOVE_RIGHT_SIMPLENAME**Strip the Simple Name on the Right from the Full Name**

The Remove the Simple Name on the Right from the Full Name routine removes the rightmost simple name from an opaque full name. It returns both the simple name stripped and the new full name that results from the operation.

Format**DNS\$REMOVE_RIGHT_SIMPLENAME**

fullname [,resulting-fullname] [,resulting-fullname-length] [,resulting-simplename]
[,resulting-simplename-length]

Returns

VMS Usage: cond_value
type: longword (unsigned)
access: write only
mechanism: by value

Arguments**fullname**

VMS Usage: char_string
type: character string
access: read only
mechanism: by descriptor

The opaque full name to strip. The **fullname** argument is the address of a descriptor pointing to the opaque full name to strip. When the opaque full name does not contain any simple names, the routine returns a value of 0 in **cond_value**.

resulting-fullname

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

The opaque full name resulting from the operation. The **resulting-fullname** argument is the address of a descriptor pointing to a buffer containing the resulting opaque full name. This buffer can be the same as the buffer referred to by the **fullname** argument; however, the descriptors must be separate.

resulting-fullname-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

DNS\$ Run-Time Routines

DNS\$REMOVE_RIGHT_SIMPLENAME

The length of the full name returned in **resulting-fullname**. The **resulting-fullname-length** argument is the address of a word that receives the length of the full name returned in **resulting-fullname**.

resulting-simplename

VMS Usage: char_string
type: character string
access: write only
mechanism: by descriptor

A buffer containing the opaque simple name stripped from **fullname**. The **resulting-simplename** argument is the address of a descriptor pointing to the buffer.

resulting-simplename-length

VMS Usage: word_unsigned
type: word (unsigned)
access: write only
mechanism: by reference

The length of the simple name. The **resulting-simplename-length** argument is the address of a word receiving the length of the simple name returned in **resulting-simplename**.

Description

DNS\$REMOVE_RIGHT_SIMPLENAME removes the rightmost simple name from an opaque full name. When **resulting-fullname** is nonzero, the full name resulting from the removal of the rightmost simple name is returned in that buffer. When **resulting-simplename** is nonzero, the simple name that was stripped from **fullname** is returned in that buffer. The namespace name is not stripped from the full name; only simple names are affected.

Condition Values Returned

SS\$_NORMAL	Normal successful completion.
DNS\$_INVALIDNAME	The name to be converted is not a valid DNS name.
-1	Error stripping name.

C.2.10 Starting the DNS Clerk

The Distributed Name Service (DNS) facility is a product consisting of two modules: a clerk and a server. The DNS clerk is an integral part of the VMS operating system. The server is a layered product. As long as a DNS server is installed in your network, you can start the DNS clerk on your local VMS system. Then, applications can take advantage of the DNS name service.

You start the DNS clerk once DECnet is running. The DNS startup procedure defines the default DNS server, installs necessary libraries, and creates an advertiser process. Startup involves two steps:

1. Obtain the name of the default DNS server from your network administrator.
2. Execute the command procedure DNS\$CHANGE_DEF_FILE. It runs the command procedure DNS\$CLERK_STARTUP, which installs the shareable libraries and creates the advertiser process DNS\$ADVER.

To run the command procedure, enter the following command:

```
$ @SYS$STARTUP:DNS$CHANGE_DEF_FILE.COM
```

When executed, SYS\$STARTUP:DNS\$CHANGE_DEF_FILE.COM prompts for the name of the node where the DNS server is located.

Name of DNS server node?

Enter a node name, identifying the node that has the DNS server installed.

Once you have run DNS\$CHANGE_DEF_FILE.COM, you do not need to run it again unless you want to change the default DNS server or the default namespace. DNS\$CHANGE_DEF_FILE.COM copies a configuration file to SYS\$SYSTEM that is called DNS\$DEFAULT_FILE.DAT. It lists the name of the namespace currently being used as a default.

You must add the following line to SYS\$MANAGER:SYSTARTUP.COM after the line that starts DECNET: @SYS\$STARTUP:DNS\$CLERK_STARTUP.COM. When the system boots, this line installs the DNS clerk images and starts the advertiser.

C.2.11 DECnet Event Messages

The following are DECnet event messages sent by the Distributed Name Service clerk. For a complete list of DECnet event messages, see the *VMS Network Control Program Manual*.

353.5 DNS Clerk Unable to Communicate with Server

The DNS clerk was unable to communicate with a DNS server. This message displays the name of the clearinghouse where the communication failed, the node on which the DNS server servicing the clearinghouse exists, and the reason why the communication failed, which might be any of the following:

- Unknown clearinghouse

The requested clearinghouse is not serviced by the DNS server that was contacted. This can happen when the cache maintained by the local DNS clerk contains outdated information for a directory.

- Clearinghouse down

A DNS server is unable to service a request because the clearinghouse is not operational (stopped state).

VMS Version 5.3 Features

DECnet Event Messages

- Wrong state

A DNS server is unable to service a request because the clearinghouse is currently starting up or shutting down.

- Data corruption

A DNS server is unable to service the request because the clearinghouse file has been corrupted.

- No communication

A network error occurred on the local system or on the system containing the DNS server. The local VMS error is displayed as a part of this message.

353.20 Local DNS Advertiser Error

This event communicates errors that are local to the DNS Advertiser (DNS\$ADVER). All these errors have the prefix ADV and are generated when the DNS Advertiser encounters an error that prevents proper operation of the process. Exact errors are listed in the *VMS System Messages and Recovery Procedures Reference Manual*.

VMS Version 5.2 Features

This appendix describes features that were new to Version 5.2 of the VMS operating system but are not yet documented in other printed manuals.

D.1 VMS Version 5.2 System Management Features

The following sections describe enhancements to these components of the VMS operating system:

- System Generation Utility (SYSGEN)
- NETCONFIG.COM
- Backup Utility (BACKUP)

D.1.1 System Generation Utility (SYSGEN)

The VMS Version 5.2 System Generation Utility (SYSGEN) contains the following new command and parameter:

- DEINSTALL command
- ERLBUFFERPAGES parameter

D.1.1.1 DEINSTALL Command Description

DEINSTALL removes or “deinstalls” system page files and system swap files. Any file that is installed with the INSTALL command can be removed with the DEINSTALL command.

Use of the DEINSTALL command requires the CMKRNL privilege.

Format

DEINSTALL filespec

Parameter

filespec

Specifies the name of the page or swap file. The default file type is SYS.

Qualifiers

/ALL

Deinstalls all page and swap files currently installed on the system. This command is most useful during an orderly system shutdown procedure where all disk volumes are being dismounted.

/INDEX=n

Deinstalls a page or swap file specified by the page file index. The page file index is presented in the SHOW MEMORY/FILES/FULL display as “Paging File Number.”

/PAGEFILE

Specifies that the file to be deinstalled is a page file.

VMS Version 5.2 Features

DEINSTALL Command Description

/SWAPFILE

Specifies that the file to be deinstalled is a swap file.

Example

```
SYSGEN> DEINSTALL SYS$SYSTEM:PAGEFILE.SYS/PAGEFILE
```

The command in this example deinstalls the system page file.

D.1.1.2 ERLBUFFERPAGES Parameter

The ERLBUFFERPAGES parameter specifies the number of pages of memory to allocate for each buffer requested by the ERRORLOGBUFFERS parameter. The ERLBUFFERPAGES parameter has a default value of 2 pages and a maximum value of 32 pages. The default value of 2 pages consists of one page for each buffer greater than the previous buffer size.

D.1.2 NETCONFIG.COM Security Enhancements

In VMS Version 5.2, the DECnet network configuration command procedure NETCONFIG.COM has been enhanced to provide several options for restricting default access. A new command procedure for existing networked systems, NETCONFIG_UPDATE.COM, described in Section D.1.3, has been created for the same purpose.

You can plan the appropriate level of default access for your system and implement that plan by responding to a few questions posed by NETCONFIG.COM. NETCONFIG.COM then automatically records your choices in the UAF (user authorization file) and in the network configuration database.

Previously, NETCONFIG.COM created one default account named DECNET. That account provided default access to all network objects and applications that were not restricted by other forms of access control (for example, proxy accounts and access control lists). If you chose to limit default access, it was necessary to manually enter all the appropriate commands in the UAF, using the Authorize Utility, and in the network configuration database, using NCP commands.

D.1.2.1 Default Access Options

NETCONFIG.COM provides two different ways to limit default access. The most restrictive form is to not create the default DECnet account but to grant default access for certain system objects by creating a default account for each one that you want to use. Using NETCONFIG.COM, you can create an account for one or more of the following network objects:

- MAIL
- File access listener (FAL)
- PHONE
- Network management listener (NML)
- Loopback mirror (MIRROR)
- VMS Performance Monitor (VPM)

The second, less restrictive form of default access is to create a default DECnet account but to disable default access to user-written programs and procedures (also known as TASK objects). Default access for system objects is still enabled.

You can still create an unrestricted default DECnet account that includes default access to TASK objects. This type of access is suitable for systems with low security requirements. To do so, you must override the defaults provided by NETCONFIG.COM.

Note

If you do not create the default DECnet account, you must create a default account for MAIL and VPM, if you want to use them. The same is true for the MIRROR object if you want to use the User Environment Test Package (UETP) to test the network connection.

FAL, if enabled by the default DECnet account or a separate default account, makes a system vulnerable to unauthorized access. Digital advises against creating a default account for FAL. Note, however, that when you do *not* use FAL with a default account, remote file requests must include explicit file access control information or the local system manager must set up proxy access for remote users. Consider an example with a local node (ETHQKE) and a remote node (MISHA) with no default account. Entering the command \$DIR MISHA:: from node ETHQKE produces the following messages:

```
%DIRECT-E-OPENIN, error opening MISHA::*.*;* as input
-RMS-E-FND, ACP file or directory lookup failed
-SYSTEM-F-INVLOGIN, login information invalid at remote node
```

However, you can access node MISHA by entering the command \$DIR MISHA"Username Password":: from node ETHQKE.

The system manager could also, by using AUTHORIZE, enable proxy access for node ETHQKE by adding REMOTE_USER_FOO, as shown in the following example:

```
$ SET DEF SYS$SYSTEM
$ RUN AUTHORIZE
UAF> ADD/PROXY/DEFAULT ETHQKE::REMOTE_USER_FOO LOCAL_USER
UAF> EXIT
```

Entering the command \$ DIR MISHA:: from node ETHQKE would then give user ETHQKE::REMOTE_USER_FOO access to remote node MISHA by proxy; MISHA then associates this account with the account LOCAL_USER on node MISHA.

The MIRROR object is used for loopback testing. To test your network connection with VAX UETP, you must create a default account for the MIRROR object, if you did not create the default DECnet account.

The VPM object is used by the Monitor Utility in VAXcluster configurations to obtain performance information about VAXcluster members. If your system is a member of a VAXcluster and the cluster manager wants to use the Monitor Utility to collect such information, you must create a default account for the VPM object, if you did not create the default DECnet account.

D.1.2.2 Security Benefits

The DECnet account provides default access for all incoming links (unless this access is overridden by other forms of access control). However, default accounts for any of the system objects named in the NETCONFIG.COM procedure limit access to these objects. Default accounts for selected objects, when used with other system security facilities, enable a system or network manager to monitor these accounts and to detect unauthorized access.

For each default account that you create, NETCONFIG.COM generates a password and registers it in your network configuration database. Such system-generated passwords are more secure than the passwords that users typically create.

VMS Version 5.2 Features

Questions Posed by NETCONFIG.COM

D.1.2.3 Questions Posed by NETCONFIG.COM

NETCONFIG.COM poses the following questions (the responses in brackets are the default values):

Do you want a default DECnet account? [NO]:

(The following question is asked only if you said YES to a default DECnet account.)

Do you want default access to the TASK object disabled? [YES]:

(The following questions are asked regardless of whether you said YES or NO to a default DECnet account.)

Do you want a default account for the MAIL object? [YES]:

Do you want a default account for the FAL object? [NO]:

Do you want a default account for the PHONE object? [YES]:

Do you want a default account for the NML object? [YES]:

(The following questions are asked only if you said NO to a default DECnet account.)

Do you want a default account for the MIRROR object? [YES]:

Do you want a default account for the VPM object? [YES]:

D.1.3 New NETCONFIG_UPDATE.COM for Existing Networks

NETCONFIG_UPDATE.COM is a new command procedure for existing networks that provides the same security enhancements for default access that are provided by NETCONFIG.COM (see Section D.1.2). It also provides a secondary procedure for modifying members of a VAXcluster. Both procedures are described in the following sections.

D.1.3.1 Benefits of NETCONFIG_UPDATE.COM

NETCONFIG_UPDATE.COM, unlike NETCONFIG.COM, configures default access only. It performs no other network configuration. Therefore, when you use NETCONFIG_UPDATE.COM to specify changes to default access, everything else in the configuration database remains unchanged.

NETCONFIG_UPDATE.COM, like NETCONFIG.COM, generates passwords for each account that you create for default access and for any existing default accounts that you decide to keep in your configuration database. For example, if you currently have a default account for MAIL and decide to keep it, NETCONFIG_UPDATE.COM generates a new password for it and replaces the existing password with the new one.

D.1.3.2 Using NETCONFIG_UPDATE.COM in a VAXcluster

NETCONFIG_UPDATE.COM provides a secondary procedure that updates the default access of VAXcluster members. After you run NETCONFIG_UPDATE.COM on one member of a VAXcluster, the procedure detects that it is a VAXcluster member and instructs you to run SYS\$COMMON:[SYSMGR]UPDATE_CLUSTER_MEMBERS.COM on the other VAXcluster members. This secondary procedure will modify the default access of each VAXcluster member exactly as you modified that of the first member.

With the SYSMAN Utility (see the *VMS SYSMAN Utility Manual*), you can use the SET ENVIRONMENT/CLUSTER command to execute this secondary procedure only once. The default access of all the remaining VAXcluster members will be updated automatically.

D.1.4 Backup Utility (BACKUP)

This section describes the following new Backup Utility (BACKUP) features:

- Performance enhancements that cause BACKUP save and copy operations to complete more quickly on systems that are configured correctly
- Faster cyclic redundancy checking (CRC) emulation for processors that emulate CRC in software, resulting in a significant performance enhancement for BACKUP on these processors
- Support for the control character Ctrl/T, which returns information about the online or standalone BACKUP operation in progress

D.1.4.1 Performance Enhancements

Version 5.2 of the Backup Utility includes a new method of scanning files on the input disk. This new file scanning method results in faster save and copy operations on systems that are configured correctly. (It does not improve BACKUP's performance during restore, compare, verify, or list operations, however.) Prior to Version 5.2, disk head movement on the input disk constrained the speed at which BACKUP could save or copy files.

To take full advantage of the new BACKUP file scanning method, you must change the values of certain user authorization file (UAF) and System Generation Utility (SYSGEN) parameters. Sections D.1.4.2 and D.1.4.3 specify which parameters you need to change.

D.1.4.2 Setting Up the BACKUP Account

BACKUP's new file-scanning method depends on the values of some user authorization file (UAF) parameters of the account from which you perform BACKUP operations. For example, if you perform BACKUP operations from the SYSTEM account, the UAF parameters for the SYSTEM account affect the way BACKUP performs. These UAF parameters define process quotas, which are the amounts of system resources available to a process created by the account. Digital recommends that you change the values of these UAF parameters for the account you use to perform BACKUP operations. See the *VMS Authorize Utility Manual* for more information about modifying the values of UAF parameters.

Table D-1 describes the UAF parameters that should be modified and supplies values that provide the maximum amount of resources to BACKUP. These values may not provide the best performance in all cases, however. They are intended to be general guidelines.

Note

BACKUP bases its memory consumption on the WSQUOTA value, not WSEXTENT.

VMS Version 5.2 Features

Setting Up the BACKUP Account

Table D-1 UAF Process Quotas for the BACKUP Account

UAF Parameter	Meaning	Recommended Value
WSQUOTA	The number of pages of memory the working set of the process can consume.	Equal to SYSGEN parameter WSMAX
WSEXTENT	The absolute limit of physical memory allowed to the process.	Equal to WSQUOTA
PGFLQUO	The number of pages of memory your process is allowed in the page file.	Greater than or equal to WSEXTENT
FILLM	The number of files that can be open simultaneously. BACKUP scans this number of files at one time.	Equal to the SYSGEN parameter CHANNELCNT
DIOLM	The number of direct I/O operations (usually disk operations) that can be outstanding simultaneously.	Maximum of either (3 x FILLM) or 4096
ASTLM	The number of asynchronous system traps that can be queued to the process simultaneously.	Maximum of either (3 x FILLM) or 4096
BIOLM	The maximum number of buffered I/O operations that can be outstanding simultaneously.	Less than or equal to FILLM
BYTLM	The total number of bytes of memory that can be outstanding for buffered I/O operations.	Greater than or equal to the following value: (256 x FILLM) + (6 x DIOLM)
ENQLM	The maximum number of locks that can be queued simultaneously.	Greater than FILLM

Table D-2 lists a set of UAF parameter values that may be useful for your configuration. You can choose to set the values for WSQUOTA and FILLM lower than these values under the following circumstances:

- If your disks are highly fragmented, lower values prevent BACKUP from becoming highly CPU-intensive.
- If you use BACKUP during periods of heavy system use, lower values prevent BACKUP from consuming too many system resources.

Note

If you decrease the values of UAF parameters other than WSQUOTA and FILLM, use the ratios in Table D-1 to determine appropriate values.

Alternatively, you can choose to set the values higher than these suggested values if files are stored contiguously on your disks and if you perform BACKUP operations during periods of light system use.

Table D-2 Suggested Values for UAF Process Quotas

UAF Parameter	Value
WSQUOTA	16,384
WSEXTENT	Greater than or equal to WSQUOTA
PGFLQUO	32,768
FILLM	128
DIOLM	4096
ASTLM	4096
BIOLM	128
BYTLM	65,536
ENQLM	256

After changing UAF parameters, log out of the BACKUP account and log back in, allowing the new values of the UAF parameters to be used.

D.1.4.3 Setting System Generation Utility (SYSGEN) Parameters

For the new BACKUP file-scanning method to work efficiently, the System Generation Utility (SYSGEN) parameters CHANNELCNT and WSMAX must be set to appropriate values. If the account you use to perform BACKUP operations has a FILLM value greater than the value of the SYSGEN parameter CHANNELCNT, CHANNELCNT constrains the number of files that can be opened at any one time. If the WSQUOTA value of the account is greater than the value of the SYSGEN parameter WSMAX, WSMAX constrains the number of pages of memory that the working set of the process can consume. See the *VMS System Generation Utility Manual* for more information about changing the values of SYSGEN parameters.

After changing SYSGEN parameters, shut down and reboot the system, allowing the new values of the parameters to be used.

D.1.4.4 Understanding Why the Output Device Seems Idle

Because BACKUP can scan many files at a time, it is possible that no data will be sent to the output device for up to several minutes after the save or copy operation begins. This does not indicate that BACKUP is performing slowly or that your output device is not working correctly. Depending on the values of the UAF parameters and the SYSGEN parameters, BACKUP's new file-scanning method requires a certain amount of time to become established. When the file scanning is completed, BACKUP sends the data to the output device more efficiently than it did before VMS Version 5.2.

D.1.4.5 /BUFFER_COUNT Command Qualifier Is Now Obsolete

The new file-scanning method used by BACKUP makes the command qualifier /BUFFER_COUNT obsolete. Previously, this command qualifier specified the number of buffers used in a save, compare, or restore operation to or from a tape. BACKUP now determines how many buffers to use, depending on the amount of memory available to the account performing the BACKUP operation and the number of files that account can open simultaneously.

You can still specify the /BUFFER_COUNT qualifier, however, although it has no effect. This ensures that command procedures written before VMS Version 5.2 will still operate correctly. Digital recommends that you remove the /BUFFER_COUNT qualifier from command procedures.

VMS Version 5.2 Features

Cyclic Redundancy Checking Emulation Improvements

D.1.4.6 Cyclic Redundancy Checking Emulation Improvements

The method for performing cyclic redundancy checking (CRC) emulation is now approximately 40% faster than the method used before VMS Version 5.2. This is not a BACKUP-specific improvement, but it does improve BACKUP performance on processors that emulate CRC in software. BACKUP operations that use cyclic redundancy checking (CRC is applied by default) now require significantly less time to complete on the following processors, all of which emulate CRC in software:

- MicroVAX II/VAXstation II
- MicroVAX 2000/VAXstation 2000
- MicroVAX 3200/VAXstation 3200
- MicroVAX 3500/VAXstation 3500
- MicroVAX 3600
- VAX 6200

D.1.4.7 Pressing Ctrl/T to Obtain Information About BACKUP Operations

Version 5.2 of the VMS operating system supplies an additional two lines of information when you press Ctrl/T during an online or standalone BACKUP operation. Ctrl/T interrupts execution of the BACKUP command and displays three lines of information. The first line displays information about the current process (node name, process name, system time, currently running image, elapsed CPU time, page faults, direct and buffered I/O operations, and pages in physical memory). The second line displays information about BACKUP input. The third line displays information about BACKUP output. For example, if you press Ctrl/T during a save operation, the second line displays the name of the last file scanned by BACKUP and the third line displays the save-set volume number, save-set block number, and the number of bytes in a block.

In order to use Ctrl/T, the command SET CONTROL=T must appear either in the system login command procedure or in your personal login command procedure. You can also enable Ctrl/T interactively by entering the DCL command SET CONTROL=T.

The following example shows what happens when you press Ctrl/T during a BACKUP save operation:

```
$ BACKUP/LOG DUA0:[MISHA]*.COM;* MUA0:COMPROCS.BCK/REWIND/LABEL=COMP
BACKUP-S-COPIED, copied DUA0:[MISHA]A.COM;32
BACKUP-S-COPIED, copied DUA0:[MISHA]B.COM;30
BACKUP-S-COPIED, copied DUA0:[MISHA]C.COM;16
Ctrl/T
SQUASH::MISHA 14:02:12 BACKUP CPU=00:00.18.44 PF=2101 IO=827 MEM=534
Last file scanned: DUA0:[NATASHA]D.DAT
Saveset volume: 1, saveset block: 35, (32256 byte blocks)
BACKUP-S-COPIED, copied DUA0:[MISHA]D.COM;2
BACKUP-S-COPIED, copied DUA0:[MISHA]E.COM;22
.
.
.
$
```

VMS Version 5.1 Features

This appendix describes features that were new to Version 5.1 of the VMS operating system but are not yet documented in other printed manuals.

E.1 VMS Version 5.1 Support for Compound Documents

The term **compound documents** refers to files that can contain a number of integrated components including text, graphics, and scanned images. This appendix specifically describes VMS support for using the text from DECwindows compound documents that are structured according to the Digital Document Interchange Format (DDIF) specification.

VMS commands and utilities, as well as existing application programs that accept text input, can now use the text content of DECwindows compound documents.

To support the use of DDIF text, VMS RMS has implemented a new RMS file attribute, **stored semantics**, and a DDIF-to-text **RMS extension**. The value of the stored semantics attribute is called the file **tag** and it specifies how file data is to be interpreted. When file data is to be interpreted in accordance with the DDIF specification, the appropriate file tag is DDIF. The use of file tags is limited to disk files on VMS Version 5.1 and later systems.

The DDIF-to-text RMS extension transparently extracts text from DDIF files as variable-length text records that can be accessed through the VMS RMS interface.

The enhancements made to support the reading of text from DDIF files are transparent to the user and to the application programmer. This support requires that all DDIF files in a VMS Version 5.1 environment be tagged with the DDIF file tag. DDIF files created by VMS and VMS layered products are tagged appropriately.

Section E.1.1 describes various VMS_file management commands and utilities that display, create, and preserve file tags where appropriate. Section E.1.1 also describes the way various VMS commands and utilities respond to DDIF file input. Section E.1.2 describes VMS support for DDIF files in heterogeneous computing environments. Section E.1.3 describes the changes made to the VMS RMS program interface to support the stored semantics attribute and to control access to the content of DDIF files.

E.1.1 VMS Commands and Utilities

This section describes the VMS commands and utilities that support tag maintenance by displaying, creating, and preserving the RMS file tags used with DDIF files. It also provides additional information that is relevant to the way selected VMS commands and utilities respond to DDIF file input.

VMS Version 5.1 Features

VMS Commands and Utilities

The following table lists the VMS commands and utilities that support tag maintenance:

Command/Utility	Tag Maintenance Function
DIRECTORY/FULL	Displays file tag
ANALYZE/RMS_FILE	Displays file tag
SET FILE/SEMANTICS	Creates file tag
VMS MAIL	Preserves file tag†
COPY	Preserves file tag†
BACKUP	Preserves file tag

†See text for exceptions.

Tags are made up of binary values that can be up to 64 bytes long and can be expressed using hexadecimal notation. The hexadecimal value of the DDIF tag, for example, is 2B0C8773010301. VMS permits you to assign mnemonics to tag values so that DCL commands, such as DIRECTORY/FULL, and VMS utilities, such as FDL and ANALYZE/RMS_FILE, display a mnemonic for the DDIF tag instead of the hexadecimal value. The following DCL commands have been included in the system startup command file to assign the mnemonic DDIF to the hexadecimal value for a DDIF tag:

```
$ DEFINE/TABLE=RMS$SEMANTIC_TAGS DDIF 2B0C8773010301
$ DEFINE/TABLE=RMS$SEMANTIC_OBJECTS 2B0C8773010301 DDIF
```

Using the appropriate DEFINE commands, you can assign mnemonics for other tags, including tags used with international program applications.

E.1.1.1 Displaying RMS File Tags

The DIRECTORY/FULL command and the Analyze/RMS_File Utility now display the RMS file tag for DDIF files.

E.1.1.1.1 DIRECTORY/FULL Where applicable, the DIRECTORY/FULL command now provides the value of the stored semantics tag as part of the file information returned to the user. This is the recommended method for quickly determining whether or not a file is tagged. The following display illustrates how the DIRECTORY/FULL command returns the RMS attributes for a DDIF file named X.DDIF:

```
X.DDIF;1                               File ID: (767,20658,0)
.
.
.
RMS attributes:      Stored semantics: DDIF
.
.
```


E.1.1.1.2 ANALYZE/RMS_FILE When you use the ANALYZE/RMS_FILE command to analyze a DDIF file, the utility returns the file tag as an RMS file attribute.

```
FILE HEADER
File Spec: USERD$:[TEST]X.DDIF;1
```

```
.
.
.   Stored semantics: DDIF
.
.
```

One ANALYZE/RMS_FILE command option is to create an output FDL file that reflects the results of the analysis, using the following format:

ANALYZE/RMS_FILE/FDL filespec

When you use this option for analyzing a tagged file, the output FDL file includes the file tag as a secondary attribute to the FILE primary attribute. This is illustrated in the following FDL file excerpt:

```
IDENT      " 9-JUN-1989 13:27:30 VAX/VMS ANALYZE/RMS_FILE Utility"
.
.
.
SYSTEM
FILE      SOURCE          VMS
          ALLOCATION      3
.
.
.   STORED_SEMANTICS      %X'2B0C8773010301' ! DDIF
.
.
```

E.1.1.2 Creating RMS File Tags

The CDA\$CREATE_FILE routine in the Compound Document Architecture toolkit creates and tags DDIF files. However, you might encounter a DDIF file that was created without a file tag or a DDIF file whose file tag was not preserved during file processing.

The DCL command SET FILE provides a qualifier, /[NO]SEMANTICS, that permits you to tag a DDIF file through the DCL interface for VMS Version 5.1 or later systems. You can also use the qualifier to change a tag or to remove a tag from a file.

The following command line tags the file X.DDIF as a DDIF file by assigning the appropriate value to the /SEMANTICS qualifier:

```
$ SET FILE X.DDIF/SEMANTICS=DDIF
```

See Section E.1.1 for information about how to use logical name tables to assign a mnemonic to a tag.

A subsequent DIRECTORY/FULL command displays the following line as part of the file header:

VMS Version 5.1 Features

Creating RMS File Tags

RMS attributes: Stored semantics: DDIF

The next example illustrates how to use the SET FILE command to delete an RMS file tag:

```
$ SET FILE X.DDIF/NOSEMANTICS
```

E.1.1.3 Preserving RMS File Tags and DDIF Semantics

The COPY command and the VMS Mail Utility preserve RMS file tags and DDIF semantics when you copy or mail a DDIF file on a VMS Version 5.1 or later system, except for conditions described in Sections E.1.2.2, E.1.2.3, and E.1.2.4.

The Backup Utility always preserves file tags and semantics when you back up a DDIF file to magnetic tape.

E.1.1.3.1 COPY Command This section describes the results of using the COPY command with DDIF files for various operations.

When you use the COPY command to copy a DDIF file to a disk on a VMS Version 5.1 or later system, VMS RMS preserves the DDIF tag and the DDIF semantics of the input file in the output file.

When you use the COPY command to copy a DDIF file to a nondisk device on a VMS Version 5.1 or later system, VMS RMS does *not* preserve the DDIF tag or the DDIF semantics of the input file in the output file. Instead, VMS RMS writes the text from the input file to the output file as variable-length records.

When you copy two or more DDIF and text files in any combination to a single output file, the output file takes the characteristics of the first input file, as shown in the following examples:

1. In this example, the first input file is a text file, so the output file (FOO.TXT) contains variable-length text records from X.TXT, Y.DDIF, and Z.TXT but does not include the DDIF tag from Y.DDIF.

```
$ COPY X.TXT,Y.DDIF,Z.TXT FOO.TXT
```

2. In this example, the first input file (A.DDIF) is a DDIF file, so the output file (FOO.DDIF) includes the DDIF tag as well as the DDIF semantics from A.DDIF. The attempt to copy the text input file (Z.TXT) fails because there is no text-to-DDIF RMS extension, but the contents of B.DDIF and C.DDIF are copied to the output file. However, the output file has no practical use because, as a result of the way DDIF files are structured, only the data from the first input file (A.DDIF) is accessible in the output file.

```
$ COPY A.DDIF,B.DDIF,Z.TXT,C.DDIF FOO.DDIF
```

3. In this example, the first input file (A.DDIF) is a DDIF file, so the output file (FOO.DDIF) includes the DDIF tag as well as the contents of A.DDIF. FOO.DDIF also includes the contents of B.DDIF and C.DDIF. Again, however, the output file has no practical use because, as a result of the way DDIF files are structured, only the data from the first input file (A.DDIF) is accessible in the output file.

```
$ COPY A.DDIF,B.DDIF,C.DDIF FOO.DDIF
```


E.1.1.3.2 VMS Mail Utility The VMS Mail Utility preserves the DDIF file tag when DDIF files are mailed between VMS Version 5.1 or later systems. The VMS Mail Utility also preserves the DDIF file tag when you create an output file on a VMS Version 5.1 or later system using the EXTRACT command.

When you read a mail message that is a DDIF file, the VMS Mail Utility outputs only the text portion of the file. Similarly, if you edit a DDIF mail file, you can access only the file text; the output file is a text file that can no longer be used as a DDIF file. However, if you forward a message that consists of a DDIF file, the VMS Mail Utility sends the entire DDIF file, including DDIF semantics and the DDIF tag, to the addressee.

E.1.1.4 APPEND Command

This section describes what happens when you attempt to use the APPEND command with DDIF and text files.

In the first example, the APPEND command appends a DDIF file to a text file:

```
$ APPEND X.DDIF Y.TXT
```

The output file, Y.TXT, contains its original text records as well as text from the input file, X.DDIF, reformatted as variable-length text records.

In the next example, the APPEND command appends a DDIF file to another DDIF file:

```
$ APPEND X.DDIF Y.DDIF
```

The output file, Y.DDIF, contains the DDIF tag, the original contents of Y.DDIF, and the contents of X.DDIF. However, the portion of the file that contains X.DDIF is not accessible because of the way DDIF files are structured.

In the final example, the APPEND command attempts to append a text file to a DDIF file:

```
$ APPEND X.TXT Y.DDIF
```

This append operation fails because there is no text-to-DDIF RMS extension.

E.1.2 DDIF Support in a Heterogeneous Environment

This section describes the implementation of DDIF support in two heterogeneous environments. The first heterogeneous environment includes VMS Version 5.1 or later systems and non-VMS systems. The second heterogeneous environment includes VMS Version 5.1 or earlier systems.

E.1.2.1 EXCHANGE/NETWORK Command

A new DCL command, EXCHANGE/NETWORK, has been created to support the transfer of files between VMS systems and non-VMS systems that do not support VMS file types. The EXCHANGE/NETWORK command transfers files in either record mode or block mode but can be used only when both systems support DECnet file transfers.

To interactively tag a DDIF file and transfer the file between a non-VMS operating system and a VMS Version 5.1 or later system, do the following:

1. Create the following file, assigning it the name DDIF.FDL:

FILE	ORGANIZATION	sequential
	STORED_SEMANTICS	DDIF

VMS Version 5.1 Features

EXCHANGE/NETWORK Command

RECORD		
CARRIAGE_CONTROL		none
FORMAT		fixed
SIZE		512

2. To transfer the desired file, enter the EXCHANGE/NETWORK command, using the following format:

EXCHANGE/NETWORK/FDL=DDIF.FDL input_filespec output_filespec

See Section E.2 for more information about the EXCHANGE/NETWORK command.

E.1.2.2 COPY Command

If you use the COPY command to copy tagged DDIF files to systems other than VMS Version 5.1 systems from a VMS Version 5.1 system, the results will vary depending on the target system:

- If the target system is a non-VMS system, the file is copied, but the DDIF tag is not preserved.
- If the target system is a VMS Version 5.1 or earlier system, the copy operation fails with the VMS RMS error message RMS\$_SUPPORT, network operation not supported, and a secondary error message of RMS\$_SEMANTICS, inconsistent usage of RMS Semantics. Error messages similar to the following will appear:

```
%COPY-E-OPENOUT, error opening PWEDGE::[]TRY.DDIF;1 as output
-RMS-F-SUPPORT, network operation not supported
-RMS-E-SEMANTICS, inconsistent usage of RMS Semantics
%COPY-W-NOTCOPIED, ABCD4:[DAVIDS]TRY.DDIF;1 not copied
```

- If the target system is a cluster alias for a mixed-version cluster containing Version 5.1 or earlier systems, the result of the copy operation depends on whether the cluster node that actually handles the request is a Version 5.1 or earlier system.
- If you use the COPY command to copy tagged DDIF files from Version 5.1 or later systems to earlier systems while on an earlier system, the copy operation will fail with the error message RMS\$_NET, network operation failed at remote node, and with a DAP status code of 16F, inconsistent usage of RMS Semantics. Error messages similar to the following will appear:

```
%COPY-E-OPENIN, error opening ARC"davids password"::ABCD4:[DAVIDS]TRY.DDIF;1 as
input
-RMS-F-NET, network operation failed at remote node; DAP code = 01F7516F
%COPY-W-NOTCOPIED, ARC"davids password"::ABCD4:[DAVIDS]TRY.DDIF;1 not copied
PWEDGE$
```

E.1.2.3 VMS Mail Utility

If you try to send mail messages containing DDIF files to non-VMS systems that do not support tagged files, the VMS Mail Utility returns the NOACCEPTMSG error message, indicating that the remote node cannot accept the message format.

Similarly, the VMS Mail Utility does not support the mailing of DDIF files to systems earlier than Version 5.1. As with non-VMS systems, the VMS Mail Utility returns the NOACCEPTMSG error message for systems earlier than Version 5.1, indicating that the remote node cannot accept the message format.

E.1.2.4 DDIF File Access Within a Mixed-Version Cluster

In a cluster that contains both Version 5.1 or earlier systems, operations on DDIF files from systems earlier than Version 5.1 will cause inconsistent behavior. Records read from DDIF files on systems earlier than Version 5.1 will be fixed-length 512-byte records, which contain DDIF control information in addition to the text context. Thus, typing a DDIF file on a system earlier than Version 5.1 does not produce readable text.

Copying a DDIF file using a system earlier than Version 5.1 will not preserve the DDIF tag on the output file, which will cause problems in later access to the new file from a Version 5.1 or later system.

However, using the Backup Utility from systems earlier than Version 5.1 will create a correct backup of DDIF files and will properly restore DDIF files from BACKUP save sets.

E.1.3 VMS RMS Interface Changes

This section provides details about the changes made to the VMS RMS interface that support access to text in VMS DECwindows DDIF files. It includes information related to tagging files and accessing tagged files through the VMS RMS interface. The section also describes how tags are preserved at the VMS RMS interface.

E.1.3.1 Programming Interface for File Tagging

This section focuses on the use of the DDIF tag for supporting VMS DECwindows files, although VMS RMS also supports file tagging for other compound document data formats.

You can tag a file from the VMS RMS interface by using the \$CREATE service in conjunction with a new extended attribute block (XAB) called the item XAB (\$XABITM). The \$XABITM macro is a general-purpose macro that was added to the RMS interface to support several Version 5.0 features. Tagged file support involves the use of the two item codes shown in Table E-1.

Table E-1 Tag Support Item Codes

Item	Buffer Size	Function
XAB\$_STORED_SEMANTICS	64 bytes maximum	Defines the file semantics established when the file is created
XAB\$_ACCESS_SEMANTICS	64 bytes maximum	Defines the file semantics desired by the accessing program

The entries XAB\$_STORED_SEMANTICS and XAB\$_ACCESS_SEMANTICS in the item list can represent either a control (set) function or a monitor (sense) function that can be passed to VMS RMS from the application program by way of the RMS interface.

The symbolic value XAB\$K_SEMANTICS_MAX_LEN represents the tag length. This value can be used to allocate buffer space for sensing and setting stored semantics for the DDIF file.

Within any one \$XABITM, you can activate either the set function or the sense function for the XAB\$_STORED_SEMANTICS and XAB\$_ACCESS_SEMANTICS items, because a common field (XAB\$B_MODE) determines which function is active. If you want to activate both the set function and the sense function for

VMS Version 5.1 Features

Programming Interface for File Tagging

either or both items, you must use two \$XABITM control blocks, one for setting the functions and one for sensing the functions.

Each entry in the item list addressed by the \$XABITM is made up of three longwords and a longword of zero terminates the list. You can locate the item list anywhere within the readable address space for a process, but any buffers required by the related function must be located in read/write memory. If the item list is invalid, RMS returns a status of RMS\$_XAB in the RAB\$L_STS field and the address of the XAB in RAB\$L_STV.

See the *VMS Record Management Services Manual* for a detailed description of the \$XABITM macro.

Example E-1 illustrates a BLISS-32 program that tags a file through the RMS interface. The tag value shown is a 6-byte hexadecimal number representing the code for the DDIF tag. The VMS RMS program interface accepts only hexadecimal tag values.

To write to a tagged file without using an RMS extension, the application program must specify access semantics that match the file's stored semantics. As shown in the example, \$CREATE tags the file and \$CONNECT specifies the appropriate access semantics.

Example E-1 Tagging a File

```
MODULE TYPE$MAIN (
    IDENT = 'X-1',
    MAIN = MAIN,
    ADDRESSING_MODE (EXTERNAL=GENERAL)
) =
BEGIN
!
FORWARD ROUTINE
    MAIN : NOVALUE;                ! Main routine
!
! INCLUDE FILES:
!
LIBRARY 'SYS$LIBRARY:LIB';
OWN
    NAM          : $NAM(),
    RETLEN,
    DDIF_TAG      : BLOCK[ 7, BYTE]
                    INITIAL( BYTE(%X'2B',%X'0C',%X'87',%X'73',%X'01',%X'03',%X'01')),
    FAB_XABITM    :
                    $xabitm
                    ( itemlist=
                      $ITMLST_UPLIT
                      (
                        (ITMCOD=XAB$_STORED_SEMANTICS,
                          BUFADR=DDIF_TAG,
                          BUFSIZ=%ALLOCATION(DDIF_TAG))
                      ),
                      mode = SETMODE),
    RAB_XABITM    :
                    $xabitm
                    ( itemlist=
                      $ITMLST_UPLIT
                      (
                        (ITMCOD=XAB$_ACCESS_SEMANTICS,
                          BUFADR=DDIF_TAG,
```

(continued on next page)

Example E-1 (Cont.) Tagging a File

```

                                BUFSIZ=%ALLOCATION(DDIF_TAG))
                                ),
                                mode = SETMODE),
FAB      : $FAB( fnm = 'TAGGED-FILE.TEST',
                                nam = NAM,
                                mrs = 512,
                                rfm = FIX,
                                fac = <GET,PUT,UPD>,
                                xab = FAB_XABITM),
REC      : BLOCK[512,BYTE],
STATUS,
RAB      : $RAB( xab = RAB_XABITM,
                                fab = FAB,
                                rsz = 512,
                                rbf = REC,
                                usz = 512,
                                ubf = REC),
DESC      : BLOCK[8,BYTE] INITIAL(0);
ROUTINE MAIN : NOVALUE =
BEGIN
STATUS = $CREATE( FAB = FAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
STATUS = $CONNECT( RAB = RAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
STATUS = $CLOSE( FAB = FAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
END;
END
ELUDOM

```

E.1.3.2 Accessing a Tagged File

This section provides details of how VMS RMS handles access to tagged files at the program level. When a program accesses a tagged file, VMS RMS must determine whether and when to associate an RMS extension with the access. This is important to the programmer because an RMS extension can change the attributes of the accessed file.

For example, a DDIF file is stored as a sequentially organized file having 512-byte, fixed-length records. If the DDIF-to-text RMS extension is used to extract text from a DDIF file, the accessed file appears as a sequentially organized file having variable-length records with a maximum record size of 2048 bytes and an implicit carriage return.

One consideration in determining whether an access requires the RMS extension is the type of access (FAB\$B_FAC). When an application program opens a file through the VMS RMS program interface, it must specify if it will be doing record I/O (default), block I/O (BIO), or mixed I/O (BRO), where the program has the option of using either block I/O or record I/O for each access. For example, if block I/O operations are specified, VMS RMS does not associate the RMS extension with the file access.

VMS Version 5.1 Features

Accessing a Tagged File

Another consideration is whether the program senses the tag when it opens a file. If the program does not sense the tag when it opens a DDIF file for record access, VMS RMS associates the RMS extension during the \$OPEN and returns the file attributes that have been modified by the extension.

The final consideration is the access semantics the program specifies and the file's stored semantics (tag). If the program specifies block I/O (FAB\$V_BIO) operations, RMS does not associate the RMS extension and the \$OPEN service returns the file's stored attributes to the accessing program regardless of whether the program senses tags.

E.1.3.2.1 File Accesses That Do Not Sense Tags This section describes what happens when a program does not use the XABITM to sense a tag when it opens a file.

When a program opens a DDIF file for record operations and does not sense the tag, VMS RMS assumes that the program wants to access text in the file. In this case, VMS RMS associates the RMS extension, which provides file attributes that correspond to record-mode access.

When a program opens a DDIF file with the FAB\$V_BRO option and does not sense the tag, any subsequent attempt to use block I/O fails. If the program specifies block I/O (FAB\$V_BIO) when it invokes the \$CONNECT service, the operation fails because the file attributes returned at \$OPEN permit record access only. Similarly, if the program specifies the FAB\$V_BRO option when it opens the file, and then specifies mixed-mode (block/record) operations by not specifying RAB\$V_BIO at \$CONNECT time, block operations such as READ and WRITE are disallowed.

E.1.3.2.2 File Accesses That Sense Tags VMS RMS does not associate the RMS extension as part of the \$OPEN service if a program opens a DDIF file and senses the stored semantics. This allows the program to specify access semantics with the \$CONNECT service. VMS RMS returns the file attributes, including the stored semantics attribute (tag value), to the program as part of the \$OPEN service.

When the program subsequently invokes the \$CONNECT service, VMS RMS uses the specified operations mode to determine its response. If the program specified FAB\$V_BRO with the \$OPEN service and then specifies block I/O (RAB\$V_BIO) when it invokes the \$CONNECT service, VMS RMS does not associate the RMS extension.

But if the program specifies record access or FAB\$V_BRO when it opens the file and then decides to use record I/O when it invokes the \$CONNECT service, VMS RMS compares the access semantics with the file's stored semantics to determine whether to associate the RMS extension. If the access semantics match the stored semantics, VMS RMS does not associate the RMS extension. If the access semantics do not match the stored semantics, VMS RMS associates the access with the RMS extension. In this case, the program must use the \$DISPLAY service to obtain the modified file attributes. If VMS RMS cannot find the appropriate RMS extension, the operation fails and the \$CONNECT service returns the EXTNOTFOU error message.

If the application program senses the file's stored semantics, VMS RMS allows mixed-mode operations. In this case, mixed block and record operations are permitted because the application gets record mode file attributes and data from the RMS extension and block mode file attributes and data from the file.

Example E-2 illustrates a BLISS-32 program that accesses a tagged file from an application program that does not use an RMS extension.

Example E-2 Accessing a Tagged File

```

MODULE TYPE$MAIN (
    IDENT = 'X-1',
    MAIN = MAIN,
    ADDRESSING_MODE (EXTERNAL=GENERAL)
) =

BEGIN
!
! FORWARD ROUTINE
    MAIN : NOVALUE;                                ! Main routine
!
! INCLUDE FILES:
!
LIBRARY 'SYS$LIBRARY:STARLET';
OWN
    NAM          : $NAM(),
    ITEM_BUFF    : BLOCK[ XAB$K_SEMANTICS_MAX_LEN,BYTE ],
    RETLEN,
    FAB_XABITM   :
        $xabitm
        ( itemlist=
            $ITMLST_UPLIT
            ((ITMCOD=XAB$_STORED_SEMANTICS,
              BUFADR=ITEM_BUFF,
              BUFSIZ=XAB$K_SEMANTICS_MAX_LEN,
              RETLEN=RETLEN)),
            mode = SENSEMODE),
    RAB_ITEMLIST : BLOCK[ ITM$_ITEM + 4, BYTE ],
    RAB_XABITM   : $XABITM
        ( itemlist=RAB_ITEMLIST,
          mode=SETMODE ),
    FAB          : $FAB( fnm = 'TAGGED-FILE.TEST',
                        nam = NAM,
                        fac = <GET,PUT,UPD>,
                        xab = FAB_XABITM),
    REC          : BLOCK[512,BYTE],
    STATUS,
    RAB          : $RAB( xab = RAB_XABITM,
                        fab = FAB,
                        rsz = 512,
                        rbf = REC,
                        usz = 512,
                        ubf = REC),
    DESC         : BLOCK[8,BYTE] INITIAL(0);
ROUTINE MAIN : NOVALUE =
BEGIN
STATUS = $OPEN( FAB = FAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
RAB_ITEMLIST[ ITM$_BUFSIZ ] = .RETLEN;
RAB_ITEMLIST[ ITM$_BUFADR ] = ITEM_BUFF;
RAB_ITEMLIST[ ITM$_ITMCOD ] = XAB$_ACCESS_SEMANTICS;
STATUS = $CONNECT( RAB = RAB );
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
STATUS = $CLOSE( FAB = FAB );

```

(continued on next page)

VMS Version 5.1 Features

Accessing a Tagged File

Example E-2 (Cont.) Accessing a Tagged File

```
IF NOT .STATUS
THEN
    SIGNAL (.STATUS);
END;
END
ELUDOM
```

E.1.3.3 Preserving Tags

To preserve the integrity of a tagged file that is being copied or transmitted, the tag must be preserved in the destination (output) file. The most efficient way to use the RMS interface for propagating tags is to open the source file (input) and sense the tag using a \$XABITM with the item code XAB\$_STORED_SEMANTICS:

```
.
.
.
ITEMLIST[ ITM$W_BUFSIZ ] = XAB$K_SEMANTICS_MAX_LEN;
ITEMLIST[ ITM$L_BUFADR ] = ITEM_BUFF;
ITEMLIST[ ITM$L_RETLEN ] = RETLEN;
ITEMLIST[ ITM$W_ITMCO ] = XAB$_STORED_SEMANTICS;
.
.
.
XABITM[ XAB$B_MODE ] = XAB$K_SENSEMODE;
STATUS = $OPEN( FAB = FAB );
.
.
.
```

Then create the destination (output) file and set the tag using a \$XABITM with the item code XAB\$_STORED_SEMANTICS:

```
.
.
.
IF .RETLEN GTR 0
THEN
    BEGIN
        ITEMLIST[ ITM$W_ITMCO ] = XAB$_STORED_SEMANTICS;
        ITEMLIST[ ITM$L_SIZE ] = .RETLEN;
        XABITM[ XAB$B_MODE ] = XAB$K_SETMODE;
    END;
STATUS = $CREATE( FAB = FAB );
.
.
.
END;
END
ELUDOM
```

E.1.4 Distributed File System Support for DDIF Tagged Files

Version 1.1 of the Distributed File System (DFS) includes limited support for DDIF tagged files. You can create and read DDIF files on a DFS device when the DFS client node is running VMS Version 5.1 or later versions. You can also use the DIRECTORY/FULL command to determine whether a DDIF file on a DFS device is tagged.

You cannot use the SET FILE/[NO]SEMANTICS command to either tag DDIF files or remove the tags from DDIF files on a DFS device. Furthermore, the Backup Utility does not preserve the DDIF tag or the DDIF stored semantics for data files on a DFS device.

E.1.5 VMS RMS Errors

Four VMS RMS error messages signal the user when the corresponding error condition exists:

- RMS\$_EXTNOTFOU
- RMS\$_SEMANTICS
- RMS\$_EXT_ERR
- RMS\$_OPNOTSUP

The RMS\$_EXTNOTFOU error message indicates that VMS RMS has not found the specified RMS extension. Verify that the file is correctly tagged, using the DIRECTORY/FULL command, and that the application program is specifying the appropriate access semantics.

VMS RMS returns the RMS\$_SEMANTICS error message when you try to create a tagged file on a remote system earlier than VMS Version 5.1 from a Version 5.1 or later system.

VMS RMS returns the RMS\$_EXT_ERR error when the DDIF RMS extension detects an inconsistency.

VMS RMS returns the RMS\$_OPNOTSUP error when the RMS DDIF extension is invoked by an RMS operation. For example, if the extension does not support write access to a DDIF file, verify that the application program is not performing record operations that modify the file.

E.2 EXCHANGE/NETWORK Command

The EXCHANGE/NETWORK command allows the VMS operating system to transfer files to or from operating systems that do not support VMS file organizations. The transfer occurs over a DECnet network communications link that connects VMS and non-VMS operating system nodes.

Using DECnet services, the EXCHANGE/NETWORK command can perform the following operations:

- Transfer files between a VMS node and a non-VMS system node
- Transfer a group of input files to a group of output files
- Transfer files between two non-VMS nodes, provided those nodes share DECnet connections with the VMS node that issues the EXCHANGE/NETWORK command

The EXCHANGE/NETWORK command imposes the following restrictions:

- Transfers of files can occur only between disk devices. (If a disk device is not the desired permanent residence for the file, you must either move the file to a disk before issuing the command or retrieve the file from a disk after the command completes.)
- The remote system must have a block size of 512 bytes, where a byte is 8 bits long.

- The nodes transferring files must support the DECnet Data Access Protocol (DAP).

The VMS Record Management Services (RMS) facility provides VMS access to records in VMS RMS files. To transfer VMS RMS files between two nodes where both nodes are VMS nodes, use one of the other DCL commands (such as COPY, APPEND, or CONVERT), as appropriate. These commands recognize RMS file organizations and are designed to ensure that RMS record structures are preserved as your files are moved.

Use the EXCHANGE/NETWORK command to transfer files between VMS nodes and non-VMS nodes when the differences in the file organizations would otherwise prevent the transfer or could lead to undesirable results. While COPY ensures that both the contents and the attributes of a replicated file are preserved, EXCHANGE/NETWORK is more flexible. EXCHANGE/NETWORK offers you explicit control of your record attributes during file transfers, with the opportunity to make a file usable on several different operating systems.

Format

EXCHANGE/NETWORK input-file-spec[,...] output-file-spec

Parameters

input-file-spec[,...]

Specifies the name of an existing file to be transferred. Wildcard characters are allowed. Use a comma (,) to indicate multiple file specifications.

output-file-spec

Specifies the name of the output file into which the input is transferred.

You must specify at least one field in the output file specification. If you omit the device or directory, your current default device and directory are used.

The EXCHANGE/NETWORK command replaces any other missing fields (file name, file type, version number) with the corresponding field of the input file specification.

EXCHANGE/NETWORK creates a new output file for every input file that you specify.

You can use the asterisk wildcard character in place of the following: file name, file type, or version number. The EXCHANGE/NETWORK command uses the corresponding field in the related input file to name the output file. You can also use the wildcard character in the output file specification to direct EXCHANGE/NETWORK to create more than one output file. For example:

```
$ EXCHANGE/NETWORK A.A,B.B MYPC::*C
```

This EXCHANGE/NETWORK command creates the files A.C and B.C at the non-VMS target node MYPC.

A more complete explanation of wildcard characters and version numbers follows in the Description section.

Description

The EXCHANGE/NETWORK command transfers files between VMS nodes and non-VMS nodes connected to the same DECnet network. If the non-VMS system does not support VMS file organizations, EXCHANGE/NETWORK can modify or discard file and record attributes during the transfer. However, if the target system is a VMS node, you have the option of applying new file and record attributes to the output file by supplying a File Definition Language (FDL) file, as described later in this section. EXCHANGE/NETWORK provides a number of defaults to handle the majority of transfers properly. However, in some situations, you need to know your file or record format requirements at both nodes.

VMS File and Record Attributes

All RMS files in the VMS environment include stored information, known as the file and record attributes, to describe the file and record characteristics. File attributes consist of items such as file organization, file protection, and file allocation information. Record attributes consist of items such as the record format, record size, key definitions for indexed files, and carriage control information. These attributes define the data format and access methods for the VMS RMS facility.

Non-VMS operating systems that do not support VMS file organizations have no means of storing file and record attributes with their files. Transferring a VMS file to a non-VMS system that is unable to store and handle file and record attributes can result in most of this information being discarded. Removing these attributes from a file can render it useless if it must be returned to the VMS system.

Transferring Files to VMS Nodes

When you transfer files to a VMS system from a non-VMS system, the files typically assume default file and record attributes. However, you can specify the attributes that you want the file to acquire in a File Definition Language (FDL) file. If you specify an FDL file with the /FDL qualifier, the FDL file determines the characteristics of the output file. This feature is useful in establishing compatible file and record attributes when you transfer a file from a non-VMS system to a VMS system. However, when you use an FDL file, you also assume responsibility for determining the required characteristics.

See the *VMS File Definition Language Facility Manual* for more information about FDL files.

Transferring Files to Non-VMS Nodes

EXCHANGE/NETWORK discards file and record attributes associated with a VMS file during a transfer to a non-VMS system that does not support VMS file organizations. Be aware that the loss of file and record attributes in the transfer can render the output file useless for many applications.

Selecting Transfer Modes

The EXCHANGE/NETWORK command has four transfer mode options: AUTOMATIC, BLOCK, RECORD, and CONVERT. For most file transfers, AUTOMATIC is sufficient. The AUTOMATIC transfer mode option allows EXCHANGE/NETWORK to transfer files using either block or record I/O. The selection is based on the input file organization and the operating systems involved.

Selecting the BLOCK transfer mode option forces EXCHANGE/NETWORK to open both the input and output files for block I/O access. The input file is then transferred to the output file block by block. Use this transfer mode when you transfer executable images. It is also useful when you must preserve a file's content exactly, which is a common requirement when you store files temporarily on another system or when cooperating applications exist on the systems.

Selecting the RECORD transfer mode option forces EXCHANGE /NETWORK to open both the input file and output file for record I/O access. The input file is then transferred to the output file record by record. This transfer mode is primarily used for transferring text files.

Selecting the CONVERT transfer mode option forces EXCHANGE /NETWORK to open the input file for RECORD access and the output file for BLOCK access. Records are then read in from the input file, packed into blocks, and written to the output file. This transfer mode is primarily used for transferring files with no implied carriage control. For example, to transfer a file created with Digital Standard Runoff (DSR) to a DECNET-DOS system, you must use the CONVERT transfer mode option. To transfer the resultant output file back to a VMS node, use the AUTOMATIC transfer mode option.

Wildcard Characters

Wildcard characters are permitted in the file specifications and follow the behavior typical of other VMS commands with respect to the VMS node.

When more than one input file is specified but wildcards are not specified in the output file specification, the first input file is copied to the output file, and each subsequent input file is transferred and given a higher version number of the same output file name. Note that the files are not concatenated into a single output file. Also note that when you transfer files to foreign systems that do not support version numbers, only one output file results, and it is the last input file.

To create multiple output files, specify multiple input files and use at least one of the following:

- An asterisk wildcard character in the output file name, file type, or version number field
- Only a node name, a device name, or a directory specification as the output file specification

When you create multiple output files, EXCHANGE/NETWORK uses the corresponding field from each input file in the output file name.

Use the /LOG qualifier when you specify multiple input and output files to verify that the files were copied as you intended.

Version Numbers

The following guidelines apply when the target node file formats accept version numbers.

If no version numbers are specified for input and output files, the EXCHANGE /NETWORK command (by default) assigns a version number to the output files that is either of the following:

- The version number of the input file
- A version number one greater than the highest version number of an existing file with the same file name and file type

When the output file version number is specified by an asterisk wildcard character, the EXCHANGE/NETWORK command uses the version numbers of the associated input files as the version numbers of the output files.

If the output file specification has an explicit version number, the EXCHANGE/NETWORK command normally uses that number for the output file specification. However, if an equal or higher version of the output file already exists, no warning message is issued, the file is copied, and the version number is set to a value one greater than the highest version number already existing.

File Protection and Creation/Revision Dates

The EXCHANGE/NETWORK command treats an output file as a new file when any portion of the output file name is explicitly specified. When the output node is a VMS system, the creation date for a new file is set to the current time and date. However, if the output file specification consists *only* of wildcard characters, the output file no longer qualifies as a new file and, therefore, the creation date of the input file is used. That is, if the output file specification is one of the following, the creation date becomes that of the input file: *, *.* , or *.*.*.

The revision date of the output file is always set to the current time and date; the backup date is set to zero. The output file is assigned a new expiration date. (Expiration dates are set by the file system if retention is enabled; otherwise, they are set to zero.)

When the target node is a VMS node, the protection and access control list (ACL) of the output file is determined by the following parameters, in the following order:

1. Protection of previously existing versions of the output file
2. Default protection and ACL of the output directory
3. Process default file protection

For an introduction to access control lists, see the *VMS DCL Concepts Manual*.

On VMS systems, the owner of the output file usually is the same as the creator of the output file. However, if a user with extended privileges creates the output file, the owner is either the owner of the parent directory or the owner of a previous version of the output file, if one exists.

Extended privileges include any of the following:

- SYSPRV or BYPASS
- System UIC
- GRPPRV if the owner of the parent directory (or previous version of the output file) is in the same group as the creator of the new output file
- An identifier (with the resource attribute) representing the owner of the parent directory (or previous version of the output file)

Qualifiers

/BACKUP

Modifies the time value specified with the /BEFORE or /SINCE qualifier. /BACKUP selects files according to the dates of their most recent backup. This time qualifier is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /CREATED, /EXPIRED, and /MODIFIED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/BEFORE[=time]

Selects only those files dated prior to the specified time. You can specify time as an absolute time, as a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following time qualifiers with /BEFORE to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

See the *VMS DCL Concepts Manual* for complete information about specifying time values.

/BY_OWNER[=uic]

Selects only those files whose owner user identification code (UIC) matches the specified owner UIC. The default UIC is that of the current process.

Specify the UIC using standard UIC format as described in the *VMS DCL Concepts Manual*.

/CONFIRM**/NOCONFIRM (default)**

Controls whether a request is issued before each file transfer operation to confirm that the operation should be performed on that file. The following responses are valid:

YES	NO	QUIT
TRUE	FALSE	Ctrl/Z
1	0	ALL
	Return	

You can use any combination of uppercase and lowercase letters for word responses. Word responses can be abbreviated to one or more letters (for example, T, TR, or TRU for TRUE), but these abbreviations must be unique. Affirmative answers are YES, TRUE, and 1. Negative answers are NO, FALSE, 0, and the Return key. QUIT or Ctrl/Z indicates that you want to stop processing the command at that point. When you respond with ALL, the command continues to process, but no further prompts are given. If you type a response other than one of those in the list, DCL displays an error message and redisplay the prompt.

/CREATED (default)

Modifies the time value specified with the /BEFORE or /SINCE qualifier. The /CREATED qualifier selects files based on their date of creation. This time qualifier is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /BACKUP, /EXPIRED, and /MODIFIED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/EXCLUDE=(file-spec[,...])

Excludes the specified files from the file transfer operation. You can include a directory but not a device in the file specification. Wildcard characters are allowed in the file specification. However, you cannot use relative version numbers to exclude a specific version. If you provide only one file specification, you can omit the parentheses.

/EXPIRED

Modifies the time value specified with the /BEFORE or /SINCE qualifiers. /EXPIRED selects files according to their expiration date. (The expiration date is set with the SET FILE/EXPIRATION_DATE command.) This time qualifier

is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /BACKUP, /CREATED, and /MODIFIED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/FDL=fdl-file-spec

Specifies that the output file characteristics are described in the File Definition Language (FDL) file. Use this qualifier when you require special output file characteristics. See the *VMS File Definition Language Facility Manual* for more information about FDL files.

Use of the /FDL qualifier implies that the transfer mode is block by block. However, the transfer mode you specify with the /TRANSFER_MODE qualifier prevails.

/LOG

/NOLOG (default)

Controls whether the EXCHANGE/NETWORK command displays the file specifications of each file copied.

When you use the /LOG qualifier, the EXCHANGE/NETWORK command displays the following for each copy operation: (1) the file specifications of the input and output files, and (2) the number of blocks or the number of records copied (depending on whether the file is copied on a block-by-block or record-by-record basis).

/MODIFIED

Modifies the time value specified with the /BEFORE or /SINCE qualifier. The /MODIFIED qualifier selects files according to the date on which they were last modified. This time qualifier is incompatible with the other time qualifiers that also allow you to select files according to time attributes: /BACKUP, /CREATED, and /EXPIRED. If you do not specify any of these four time qualifiers, the default is /CREATED.

/SINCE[=time]

Selects only those files dated after the specified time. You can specify time as an absolute time, a combination of absolute and delta times, or as one of the following keywords: TODAY (default), TOMORROW, or YESTERDAY. Specify one of the following time qualifiers with /SINCE to indicate the time attribute to be used as the basis for selection: /BACKUP, /CREATED (default), /EXPIRED, or /MODIFIED.

See the *VMS DCL Concepts Manual* for complete information about specifying time values.

/TRANSFER_MODE=option

Specifies the I/O method to be used in the transfer. This qualifier is useful for all file formats. You can specify any one of the following options:

Option	Function
AUTOMATIC	Allows EXCHANGE/NETWORK to determine the appropriate transfer mode.
BLOCK	Transfers block by block.

Option	Function
CONVERT[=option[,...]]	Reads records from the input file, packs them into blocks, and writes to the output file in block mode. The options determine what additional information is inserted during the transfer.
RECORD	Transfers record by record.

The AUTOMATIC transfer mode option allows EXCHANGE/NETWORK to determine the appropriate transfer mode. The default is the AUTOMATIC transfer mode.

If you explicitly select the BLOCK transfer mode option, EXCHANGE/NETWORK opens both the input and output files for block I/O. EXCHANGE/NETWORK then transfers the files block by block.

If you explicitly select the RECORD transfer mode option, EXCHANGE/NETWORK opens both the input and output files for record I/O. The target system must support record operations, and the input file must be record oriented.

If you select the CONVERT transfer mode option, EXCHANGE/NETWORK reads records in from the input file, packs them into blocks, and writes them to the output file in block mode. There are four options available with the CONVERT transfer mode to control the insertion of special characters in the records, as explained in the following paragraphs:

- CARRIAGE_CONTROL
- COUNTED
- FIXED_CONTROL
- RECORD_SEPARATOR=separator

If you specify CARRIAGE_CONTROL, any carriage control information in the input file is interpreted, expanded into actual characters, and included with each record.

If you specify COUNTED, the length of each record in bytes is included at the beginning of each record. The length includes all FIXED_CONTROL, CARRIAGE_CONTROL, and RECORD_SEPARATOR information in each record.

If you specify FIXED_CONTROL, all variable length with fixed control record (VFC) information is written to the output file as part of the data. This information follows the record length information if the COUNTED option was specified.

If you specify RECORD_SEPARATOR, a 1- or 2-byte record separator is inserted between each record. Record separator characters are the last characters in the record. The three choices for separator characters are CR for carriage return only, LF for line feed only, or CRLF for carriage return and line feed.

Examples

1. \$ EXCHANGE/NETWORK VMS_FILE.DAT FOO::FOREIGN_SYS.DAT

In this example, the EXCHANGE/NETWORK command transfers the file VMS_FILE.DAT located in the current default device and directory to the file FOREIGN_SYS.DAT on the non-VMS node FOO. Because the /TRANSFER_MODE qualifier was not explicitly specified, EXCHANGE/NETWORK automatically determines whether the transfer method should be block or record I/O.

2. \$ EXCHANGE/NETWORK/TRANSFER_MODE=BLOCK -
_ \$ FOO::FOREIGN_SYS.DAT VMS_FILE.DAT

In this example, the EXCHANGE/NETWORK command transfers the file FOREIGN_SYS.DAT from the non-VMS node FOO to the file VMS_FILE.DAT in the current default device and directory. Block I/O is specified for the transfer mode.

3. \$ EXCHANGE/NETWORK/FDL=VMS_FILE_DEFINITION.FDL -
_ \$ FOO::REMOTE_FILE.TXT VMS_FILE.DAT

In this example, the EXCHANGE/NETWORK command transfers the file REMOTE_FILE.TXT on node FOO to the file VMS_FILE.DAT. The file attributes for the output file VMS_FILE.DAT are obtained from the File Definition Language (FDL) source file VMS_FILE_DEFINITION.FDL. For more information about creating FDL files, see the *VMS File Definition Language Facility Manual*. Because the /FDL qualifier is specified and the /TRANSFER_MODE qualifier is omitted, the transfer mode uses block I/O, by default.

4. \$ EXCHANGE/NETWORK -
_ \$ /TRANSFER_MODE=CONVERT=(CARRIAGE_CONTROL,COUNTED, -
_ \$ RECORD_SEPARATOR=CRLF, FIXED_CONTROL) -
_ \$ PRINT_FILE.TXT FOO::*

In this example, the EXCHANGE/NETWORK command transfers the file PRINT_FILE.TXT from the current default device and directory to the file PRINT_FILE.TXT on the non-VMS node FOO. The use of the CONVERT option with the /TRANSFER_MODE qualifier forces the input file to be read in record by record, modified as specified by the convert options described below, and written to the output file block by block. As many records as will fit are packed into the output blocks.

The CONVERT option CARRIAGE_CONTROL specifies that carriage control information be converted to ASCII characters and inserted before the data or appended to the record, depending on whether prefix control or postfix control, or both, are used. The CONVERT option FIXED_CONTROL specifies that any fixed control information be translated to ASCII characters and inserted at the beginning of the record. The CONVERT option RECORD_SEPARATOR=CRLF appends the two specified characters, carriage return and line feed, to the end of the record. The CONVERT option COUNTED specifies that the total length of the record must be counted (once the impact of all the previous convert options have been added), and the result is to be inserted at the beginning of the record, in the first two bytes.

Abstract

The purpose of this study was to determine the effect of the type of soil on the growth of the plant. The study was conducted in a greenhouse. The plants were grown in three different types of soil: sand, loam, and clay. The results showed that the plants grown in loam soil grew the tallest and had the most leaves. The plants grown in sand soil grew the shortest and had the fewest leaves. The plants grown in clay soil grew to an intermediate height and had an intermediate number of leaves.

The study was conducted in a greenhouse. The plants were grown in three different types of soil: sand, loam, and clay. The results showed that the plants grown in loam soil grew the tallest and had the most leaves. The plants grown in sand soil grew the shortest and had the fewest leaves. The plants grown in clay soil grew to an intermediate height and had an intermediate number of leaves.

The study was conducted in a greenhouse. The plants were grown in three different types of soil: sand, loam, and clay. The results showed that the plants grown in loam soil grew the tallest and had the most leaves. The plants grown in sand soil grew the shortest and had the fewest leaves. The plants grown in clay soil grew to an intermediate height and had an intermediate number of leaves.

The study was conducted in a greenhouse. The plants were grown in three different types of soil: sand, loam, and clay. The results showed that the plants grown in loam soil grew the tallest and had the most leaves. The plants grown in sand soil grew the shortest and had the fewest leaves. The plants grown in clay soil grew to an intermediate height and had an intermediate number of leaves.

The study was conducted in a greenhouse. The plants were grown in three different types of soil: sand, loam, and clay. The results showed that the plants grown in loam soil grew the tallest and had the most leaves. The plants grown in sand soil grew the shortest and had the fewest leaves. The plants grown in clay soil grew to an intermediate height and had an intermediate number of leaves.

Index

A

Aborting a transaction, B-36, B-40
Abort reason codes, 16-1
ABORT subcommand
 with LMCP REPAIR command, B-87
Access control list
 See ACL
ACCESSIBILITY keyword
 BACKUP/IGNORE, A-3
Accounting Utility (ACCOUNTING)
 vector processing support, B-15
ACL (access control list)
 on vector capability object, B-13 to B-14
ACP-QIO interface
 movefile subfunction, 22-1
Adapter
 bus, B-59, B-60, B-61
 showing information, B-59, B-60, B-61
ADAPTER keyword
 Error Log Utility (ERROR LOG), B-62
Address
 converting to node address, C-57
 converting to node name, C-58
AGEN\$MAIL.COM, B-54
AGEN\$P1 logical name, B-52
AGEN\$P2 logical name, B-52
AGEN\$P3 logical name, B-52
AGEN\$PARAMS.REPORT, B-50
 using MAIL to send, B-54
Arithmetic exception
 debugging vector, B-19, B-28
AST (asynchronous system trap)
 vector processing synchronization issues, B-25
Asymmetric vector processing configuration, B-5
Asynchronous option
 VMS RMS support, B-110
Asynchronous system trap
 See AST
Atomic transaction
 defined, B-35
Attribute for DNS
 assigning, C-5
 enumerating, C-27
 modifying, C-30
 reading, C-31
 returning value, C-67

Attribute for DNS (Cont.)

 testing for one, C-33

 types of, C-12

Attribute_Name identifier, C-41

Attribute_Name_Str identifier, C-41

AUTOGEN

 AGEN\$PARAMS.REPORT, B-50

 command procedure for automating, B-52

 controlling size of page and swap files, B-51

 including files in MODPARAMS.DAT, B-50

 LNMSHASHTBL parameter, B-52

 logical names defined by, B-52

 LRPCOUNT parameter, B-52

 new feedback parameters, B-52

 using MAIL to send reports, B-54

 validation of parameter names, B-49

Automatic start

 See Autostart

Autostart, 5-7

 designating queues, 5-8

 disabling on a node, 5-10

 enabling on a node, 5-9

 of queue manager, 5-6

 related commands, 5-7

Autostart queues

 preventing from starting, 5-9

 restriction, 5-8

 starting, 5-7, 5-9

Availability

 of queue manager, 5-1

 of queues, 5-7, 5-8

Availability of data

 with volume shadowing, B-106

B

BACKUP command

 /MEDIA_FORMAT qualifier, B-41

/BACKUP qualifier

 EXCHANGE/NETWORK command, E-17

Backup Utility (BACKUP), A-2, A-3, D-5 to D-8

 /BUFFER_COUNT command qualifier, D-7

 cyclic redundancy checking, D-8

 /DELETE qualifier, A-2

 documentation, 1-4

 /IGNORE=LABEL_PROCESSING qualifier,

 A-3

 /IGNORE qualifier, A-3

Backup Utility (BACKUP) (Cont.)

- label processing, A-3
- /MEDIA_FORMAT qualifier, B-41
- new tape capabilities, A-1
- performance enhancements, D-5
- pressing Ctrl/T during BACKUP, D-8
- /RECORD qualifier, A-2
- /RELEASE_TAPE qualifier, A-2
- setting SYSGEN parameters to enhance performance of, D-7
- setting up BACKUP account, D-5
- summary of VMS Version 5.2 new features, D-5
- UAF parameters for BACKUP account, D-6
- Basic Linear Algebra Subroutines
 - See BLAS
- Batch log time-stamps, 3-5
 - to set prefix, 3-5
 - to verify prefix control string, 3-5
- /BEFORE qualifier
 - EXCHANGE/NETWORK command, E-18
- Big-endian
 - byte handling, A-19, A-20, A-33, A-34
 - VMEbus, A-10
- BLAS (Basic Linear Algebra Subroutines), B-7, B-21, B-23
- Boolean identifier, C-41
- /BPAGE qualifier
 - in Linker Utility, B-107, B-108
- Bugcheck
 - UBMAPEXCED, A-25, A-30
- Building dependable VMS systems, 1-4
- Byte order pattern, A-10
 - swapping, A-19, A-20
- Byte swap longword
 - for VME support, A-33
- Byte swap routine
 - for VME support, A-33, A-34
- Byte swap word
 - for VME support, A-34
- /BY_OWNER qualifier
 - EXCHANGE/NETWORK command, E-18

C

- CACHE keyword
 - Error Log Utility (ERROR LOG), B-62
- Calculator
 - hexadecimal mode, B-48
 - octal mode, B-48
- Capability
 - See also Vector capability
 - defined, B-8
- Case sensitivity
 - MACRO global symbol definitions, 19-2
 - specifying, 19-2, 19-6
- CASE_SENSITIVE option
 - in linker option file, 19-1

CDA Viewer

- in DECwindows, B-46
- new processing options
 - orientation radio box, B-47
 - Scale Factor, B-47
 - Use Bitmap Widths toggle button, B-47
 - Use Comments toggle button, B-47
 - Use Fake Trays toggle button, B-48
 - Watch Progress toggle button, B-48
- PostScript file support, B-46
- Character string
 - as arguments to linker options
 - case-sensitivity, 19-1
- Child directory
 - DNS, C-6
- CI (computer interconnect)
 - using multiple CI interfaces, B-55
 - using multiple STAR couplers, B-55
- Circuit
 - devices
 - controllers, B-105
 - DEMNA controller, B-105
 - Second Generation Ethernet Controller (SGEC), B-105
- Class_Name identifier, C-41
- Class_Name_Str identifier, C-41
- Class_Version identifier, C-41
- Clearinghouse, C-13
- Clock
 - menu bar, B-48
- Cluster-accessible tape, 7-1
- Clusterwide queue manager, 5-1
- Color name file, A-61
- COMMIT subcommand
 - with LMCP REPAIR command, B-88
- Committing a transaction, B-36, B-40
- /COMMSYNC qualifier
 - in SET TERMINAL DCL command, 18-2
- Compaction of data
 - See Data record compaction
- Compiler
 - vectorizing, B-7, B-21
- Compiling fonts
 - for DECwindows server, B-41
- Compound document
 - See also DDIF
 - defined, E-1
- Computer interconnect
 - See CI
- Condition value, C-44 to C-47
- Confidence identifier, C-41
- Confidence level, C-15
- Configuration
 - for volume shadowing, B-106
- CONFIGURATION keyword
 - Error Log Utility (ERROR LOG), B-62
- /CONFIRM qualifier

/CONFIRM qualifier (Cont.)

- EXCHANGE/NETWORK command, E-18
- Controlling data compaction, B-41
- CONVERT command
 - LMCP Utility, B-77
- CREATE command
 - LMCP Utility, B-78
- /CREATED qualifier
 - EXCHANGE/NETWORK command, E-18
- Creating a transaction log file, B-69
- Cyclic redundancy checking, D-8

D

Data

- availability with volume shadowing, B-106
- ensuring against loss, B-106
- Data compaction
 - controlling, B-41
 - volume mount, B-43
- Data record compaction, B-42
- Data structure
 - DECdtm programming routines, B-40
- DCL command
 - CONVERT/DOCUMENT
 - /MESSAGE_FILE qualifier, 2-6
 - SET FILE
 - /MOVE qualifier, 11-2
 - /NOMOVE qualifier, 11-2
 - SET TERMINAL
 - /COMMSYNC qualifier, 18-2
 - /NOCOMMSYNC qualifier, 18-2
 - summary of new and enhanced, B-41
- DDIF (Digital Document Interchange Format)
 - VMS RMS support of, E-1
- DDIF-to-text RMS extension, E-1
- Debugger, 14-1
 - support for DECthreads, 14-1
 - support for vectorized programs, B-26
- DECdtm programming routines
 - data type, B-40
- DECdtm services, 16-1 to 16-2, B-34 to B-40
 - See also Log Manager Control Program Utility
 - aborting a transaction, B-36, B-40
 - atomic transaction, B-35
 - committing a transaction, B-36, B-40
 - customizing, B-34, B-69
 - data type, B-40
 - disabling, B-34, B-69
 - log manager, B-38, B-66
 - Log Manager Control Program Utility (LMCP), B-40, B-66
 - exiting, B-75
 - invoking, B-75
 - Monitor Utility (MONITOR) support, B-40, B-95 to B-100
 - participant in a transaction, B-36, B-39, B-74
 - resource manager, B-36

DECdtm services (Cont.)

- RMS Journaling support, B-114 to B-119
- system services, B-39
- transaction identifier (TID), B-39, B-74
- transaction log file, B-38, B-66
 - creating, B-69
 - determining location, B-67
 - dumping, B-80
 - estimating file size, B-69
 - format, B-73
 - placing in alternate location, B-73
 - repairing, B-85
 - resizing, B-72
 - sample display, B-74
 - showing, B-93
- transaction manager, B-36
- transaction processing, B-34
- transaction states, B-39, B-74
- TRANSACTION_ID data type, B-40
- two-phase commit protocol, B-35, B-39
- DECnet account
 - limiting default access, D-2
- DECnet event messages, C-73
- Decompressing the system messages help library, B-44
- DECram disk
 - specifying size, 2-3
- DECthreads
 - debugger support, 14-1
- DECwindows
 - Calculator
 - hexadecimal mode, B-48
 - octal mode, B-48
 - Clock
 - menu bar, B-48
 - Mail
 - displaying PostScript files, B-48
- DECwindows CDA Viewer
 - See CDA Viewer
- DECwindows screen
 - multiscreen support, B-45
- DECwindows X11 Display Server, A-61
- DEINSTALL command, D-1
- Delta/XDelta Utility (DELTA/XDELTA)
 - support for vectorized programs, B-27
- DEMFA controller, A-52
- DEMNA controller
 - circuit name, B-105
 - line name, B-105
- Dependability
 - building into VMS systems, 1-4
- Device driver
 - VME coding conventions, A-15
- Device names
 - for VAXft 3000 system, B-57
- Digital Document Interchange Format
 - See DDIF
- DIGITAL Extended Math Library

DIGITAL Extended Math Library (Cont.)

See DXML

Direct memory access

VMEbus devices, A-12

VMEbus mapping, A-13

VMEbus map register, A-13

Directory

DNS types, C-6, C-14

enumerating in DNS, C-27

DISABLE AUTOSTART command, 5-8, 5-10

.DISABLE directive, B-22

Disabling autostart on a node, 5-10

Disabling the TP_SERVER process, B-34, B-69

Disk

repairing faulty, B-106

shadowing, B-106

Disk activity

reduced with new queue manager, 5-1

Disk space

amount needed to decompress help library,
B-44

Distributed Name Service

See DNS

DMA

See Direct memory access

DMA interface

for VMEbus device, A-35

DMA map registers

for VME, A-22, A-24, A-26

DMA routines

for VMEbus devices, A-13

DNS\$APPEND_SIMPLENAME_TO_RIGHT

routine, C-50

DNS\$COMPARE_FULLNAME routine, C-52

DNS\$COMPARE_SIMPLENAME routine, C-53

DNS\$CONCATENATE_NAME routine, C-54

DNS\$CONTEXTVARNAME item, C-43

DNS\$CONTEXTVARTIME item, C-43

DNS\$COUNT_SIMPLENAMES routine, C-56

DNS\$CVT_DNSADDRESS_TO_BINARY routine,
C-57

DNS\$CVT_DNSADDRESS_TO_NODENAME
routine, C-58

DNS\$CVT_NODENAME_TO_DNSADDRESS
routine, C-60

DNS\$CVT_TO_USERNAME_STRING routine,
C-62

DNS\$PARSE_USERNAME_STRING routine,
C-64

DNS\$REMOVE_FIRST_SET_VALUE routine,
C-67

DNS\$REMOVE_LEFT_SIMPLENAME routine,
C-69

DNS\$REMOVE_RIGHT_SIMPLENAME routine,
C-71

DNS (Distributed Name Service), C-3

child directory, C-6

clearinghouse, C-13

DNS (Distributed Name Service) (Cont.)

event messages, C-73

restrictions, C-4

root directory, C-5

system error messages, C-3

wildcards, C-9, C-20

DNS call

timeout in, C-10

DNS clerk

locating data in namespace, C-23

starting, C-73

\$DNS function code, C-26 to C-34

converting from opaque, C-30

converting opaque name, C-33

converting string name, C-31

creating an object, C-26

deleting an object, C-26

enumerating attributes, C-27

enumerating child directories, C-27

enumerating objects, C-28

enumerating soft links, C-29

modifying attributes, C-30

reading attribute, C-31

resolving soft link, C-32

testing a group, C-34

testing for attribute, C-33

\$DNS item code, C-35 to C-41

arguments, C-41 to C-42

attribute address, C-39

attribute name, C-35

attribute type, C-35

attribute value address, C-39

Boolean values, C-37

caching results, C-38

confidence level, C-36

converting names, C-36, C-37, C-38, C-40

entry type, C-36, C-37

enumerating directories, C-36

enumerating functions, C-36

enumerating objects, C-36

member name, C-38

modifying attributes, C-38

object class, C-36

object name, C-39

simple name address, C-39

soft link name, C-37

specifying groups, C-37

suppressing namespace name, C-40

target name address, C-39

testing attribute value, C-40

timeout value, C-40

UID address, C-40

version of object, C-40

wildcard, C-41

DNS name

case sensitivity, C-9

comparing, C-53

converting, C-30, C-31, C-33

- DNS name (Cont.)
 - converting full name, C-30
 - defining logicals, C-8
 - format of, C-5
 - source of, C-5
- DNS naming conventions
 - binary names, C-9
 - format, C-5
 - logical names, C-8
 - quoted names, C-9
 - syntax, C-6
 - valid characters, C-8
 - wildcards, C-9
- DNS object, C-6
 - creating, C-9 to C-11, C-26
 - deleting, C-26
 - enumerating, C-28
 - modifying, C-11 to C-13
 - reading attributes of, C-17
- DNS string name
 - converting to opaque, C-31
 - format, C-5
- \$DNS system service, C-25
 - arguments, C-25 to C-43
 - building item list, C-34
 - description, C-43 to C-44
 - format, C-25, C-43
 - function codes, C-25
 - item code identifiers, C-41
 - qualifying status, C-43
 - returns, C-25
 - status block, C-25
- \$DNSW system service, C-48
- Documentation
 - new, 1-4
- DSA disk
 - specifying preferred path, B-55
- DUMP command
 - LMCP Utility, B-80
- DWMVA adapter, A-9
 - parameter selection, A-10
- DXML (DIGITAL Extended Math Library), B-8, B-21

E

- ENABLE AUTOSTART command, 5-7
- .ENABLE directive, B-22
- Enabling autostart on a node, 5-9
- Entry_Type identifier, C-41
- Enumerate call
 - attributes, C-27
 - directories, C-27
 - objects, C-28
 - soft links, C-29
- Enum_Att_Name identifier, C-41
- ERLBUFFERPAGES parameter
 - description, D-2

- Error Log Utility (ERROR LOG)
 - qualifiers
 - /EXCLUDE
 - device class keywords, B-62
 - entry type keywords, B-62
 - /INCLUDE
 - device class keywords, B-62
 - entry type keywords, B-62
 - /NODE, B-62 to B-64
 - supported device types for VAXft 3000 systems, B-62
 - vector processing support, B-16
- Error messages, 4-2
- Ethernet, A-52
- Ethernet/820 controllers
 - circuit name, B-105
 - line name, B-105
- Event flag
 - \$DNS system service, C-25
- Event messages
 - DNS, C-73
- Exception
 - servicing vector, B-28 to B-31
- EXCHANGE/NETWORK command, E-13 to E-21
 - creating files, E-17
 - protecting files, E-17
 - qualifiers, E-17
 - selecting transfer modes, E-15
 - transferring files, E-15
 - wildcard characters, E-16
- /EXCLUDE qualifier
 - Error Log Utility (ERROR LOG)
 - device class keywords, B-62
 - entry type keywords, B-62
 - EXCHANGE/NETWORK command, E-18
- Exiting
 - LMCP, B-75
 - LMCP REPAIR command mode, B-89
- EXIT subcommand
 - with LMCP REPAIR command, B-89
- Expired-Date Suppression, B-112
- /EXPIRED qualifier
 - EXCHANGE/NETWORK command, E-18

F

- F\$ENVIRONMENT lexical function, 3-7
- F\$GETJPI lexical function, B-14 to B-15
- F\$GETQUI lexical function, 3-8 to 3-10
- F\$GETSYI lexical function, B-14 to B-15
- F\$MESSAGE lexical function, 2-7 to 2-8
- FAB\$V_ASY
 - documentation change, B-110
- Failover
 - of queue manager, 5-1
 - of queues, 5-7, 5-8
 - using shadowed disks, B-106

FAL (file access listener)
 creating a default account, D-3
 default access, D-2

Fault tolerance
 through volume shadowing, B-106

FDDI
 See Fiber distributed data interface (FDDI)

/FDL qualifier
 EXCHANGE/NETWORK command, E-19

Fiber distributed data interface (FDDI), A-1
 and Ethernet, A-52
 error code, A-5
 NCP Line Counters for, A-4
 new and changed parameters, A-54
 new type of LAN, A-53
 overview of, A-52
 programming interface, A-52

File
 copying, E-13
 creating, E-13
 transferring, E-13, E-15

File access listener
 See FAL

File Expiration Date and Time
 evaluation criteria, B-112
 usage, B-112

File protection
 EXCHANGE/NETWORK command, E-17

Files-11 On-Disk Structure Level 2 ACP, B-112

File tag
 creating, E-1
 DDIF, E-1
 disposition by COPY command, E-4
 requirement for, E-1
 stored semantics file attribute, E-1
 using, E-1

First-Order Linear Recurrence subroutines
 See FOLR subroutines

FOLR (First-Order Linear Recurrence)
 subroutines, B-7, B-21, B-23

FONT command, B-41

FORCE option
 SET VOLUME command, 2-4

FORGET subcommand
 with LMCP REPAIR command, B-90

Full name
 converting to opaque, C-31
 converting to string, C-30

Full_Name_String identifier, C-42

G

Generic queues
 restriction, 5-8

Global symbol definitions
 specifying case sensitivity, 19-2, 19-6

Group_Member identifier, C-42

H

Help
 setting up and decompressing, B-44

HELP command
 in LMCP Utility, B-84

Help library
 decompressing system messages, B-44

HELP subcommand
 with LMCP REPAIR command, B-91

HLP\$LIBRARY logical name, B-44

I

/INCLUDE qualifier
 Error Log Utility (ERROR LOG)
 device class keywords, B-62
 entry type keywords, B-62

INFORMATIONAL keyword
 Error Log Utility (ERROR LOG), B-62

INITIALIZE command
 /SIZE qualifier, 2-3

INITIALIZE/QUEUE command
 /AUTOSTART_ON qualifier, 5-7, 5-8

Interrupt
 request level, A-11
 with VME devices, A-11

IO\$_SETPRFPTH function
 specifying preferred path for DSA disks, B-56

IOC\$ALOVMEMAP_DMAN routine, A-22

IOC\$ALOVMEMAP_DMA routine, A-22

IOC\$ALOVMEMAP_PIO routine, A-28

IOC\$LOADVMEMAP_DMAN routine, A-24

IOC\$LOADVMEMAP_DMA routine, A-24

IOC\$LOADVMEMAP_PIO routine, A-29

IOC\$RELVMEMAP_DMA routine, A-26

IOC\$RELVMEMAP_PIO routine, A-31

IOC\$VME_BYTE_SWAP_LONG routine, A-33

IOC\$VME_BYTE_SWAP_WORD routine, A-34

J

JBCSYSQUE.DAT file, 5-2

Job controller
 function, 5-1
 separation from queue manager, 5-1
 starting queue manager, 5-1, 5-4

Job retention
 user-specified, 3-3

Job state
 stalled, 3-2

Journal file, 5-2
 changing location after upgrade, 5-4
 location, 5-4

L

LAD service
 bindings, 6-1
 password protection, 6-1
 write protection, 6-1

LAT
 advantages and uses, 9-6
 application programs, 9-6
 creating a VMS service, 9-4
 customizing, 9-13
 enabling outgoing connections, 9-5
 load balancing, 9-6
 managing the database size, 9-14
 modems, 9-6
 printers, 9-6
 setting up logical ports, 9-4
 terminals, 9-6

LAT\$CONFIG command procedure, 9-13

LAT\$STARTUP command procedure, 9-1, 9-13

LAT\$SYSTARTUP.COM command procedure, 9-3

LAT\$SYSTARTUP command procedure, 9-1, 9-13

LATACP process, 9-14

LAT connections
 outgoing, 9-5, 9-7

LAT Control Program (LATCP) Utility, 9-7, 9-13

LATCP
 See LAT Control Program (LATCP) Utility

LAT database
 managing size, 9-14

LAT network
 starting in SYSTARTUP_V5.COM, 9-1

LAT node
 customizing, 9-3

LAT protocol software
 starting with LAT\$STARTUP.COM, 9-1, 9-13

LAT SENSEMODE \$QIO function, 17-7

LAT service
 defined, 9-5

LAT SETMODE \$QIO function, 17-1

Lexical functions
 F\$ENVIRONMENT, 3-7
 F\$GETQUI, 3-8
 F\$MESSAGE, 2-7
 vector processing support, B-14

LIB\$GETQUI run-time library routine, 13-1

Librarian Utility (LIBRARIAN)
 using to set up online help, B-44

License
 command procedure, 10-2
 copying of a, 10-1
 moving of a, 10-1
 PAKs with reservation lists, 10-2
 registration, 10-1
 reservation list, 10-1

Line
 devices

Line

 devices (Cont.)
 controllers, B-105
 DEMNA controller, B-105
 Second Generation Ethernet Controller (SGEC), B-105

Linker options file
 case sensitivity of keyword arguments, 19-2
 CASE_SENSITIVE= option, 19-1

Linker Utility (LINK)
 /BPAGE qualifier, B-107, B-108
 CASE_SENSITIVE= option, 19-1

Little-endian
 VMEbus, A-10

LMCP
 See Log Manager Control Program Utility

LNMSHASHTBL parameter
 use with AUTOGEN feedback, B-52

Load balancing, B-55
 LAT, 9-6
 using SYSGEN parameters, B-55

LOAD_PWD_POLICY system parameter, B-57

LOAD_PWS_POLICY parameter
 in System Generation Utility (SYSGEN), B-57

LOAD_SYS_IMAGES parameter
 in System Generation Utility (SYSGEN), B-57

Local area VAXclusters, A-2

Local buffer pool
 effect on I/O performance, B-110

Local buffers
 increase in limit, B-110
 specifying number with multibuffer count
 XABITM, B-110

Lock manager limit, C-1

Log file
 See Transaction log file

Logical name, C-8
 process logical names defined by AUTOGEN, B-52
 QMAN\$MASTER, 5-4
 requirement in a VAXcluster, 5-5

Log manager, B-38, B-66

Log Manager Control Program Utility (LMCP), B-40, B-66
 command descriptions, B-76 to B-94
 CONVERT command, B-77
 CREATE command, B-78
 DUMP command, B-80
 exiting, B-75
 HELP command, B-84
 invoking, B-75
 privileges, B-75
 REPAIR command, B-85
 subcommands, B-86 to B-93
 SHOW command, B-93
 LOGOUT command
 vector processing support, B-15
 /LOG qualifier

/LOG qualifier (Cont.)

- EXCHANGE/NETWORK command, E-19
- Loopback mirror
 - See MIRROR
- LRPCOUNT parameter
 - use with AUTOGEN feedback, B-52

M

- MACRO DCL command
 - /NAMES qualifier, 19-2, 19-6
- Macros
 - VMEbus devices, A-18
- Magnetic tape
 - retensioning, 2-5
- Magnetic tape devices
 - serving within a cluster, 7-1
- MAIL
 - default access, D-3
- Mail (DECwindows)
 - displaying PostScript files, B-48
- Mailbox
 - driver, 21-1
 - function modifiers
 - IO\$M_READERCHECK, 21-2
 - IO\$M_STREAM, 21-2
 - IO\$M_WRITERCHECK, 21-2
 - wait for writer/reader function, 21-1
- Manager, queue
 - See Queue manager
- Managing the LAT database size, 9-14
- Map register
 - allocating for VME DMA, A-13
 - for VME PIO, A-13
 - loading for VME DMA, A-13
- Marginal vector consumer, B-9
 - detection of, B-13
- Master file, 5-2
 - changing location after upgrade, 5-4
- /MEDIA_FORMAT=[NO]COMPACTION qualifier, B-43
- /MEDIA_FORMAT qualifier, B-42
 - in Backup Utility (BACKUP), B-42
 - with BACKUP command, B-41
 - with MOUNT command, B-41
- Memory management, A-5
- Messages
 - facilities with new or modified system messages, 4-1
 - new system messages, 4-2
 - online help for, B-44
 - reported in a vector processing system, B-17 to B-21
- /MESSAGE_FILE qualifier
 - in CONVERT/DOCUMENT DCL command, 2-6
- MIRROR
 - default access for loopback testing, D-2

Modes

- of transferring files, E-15
- /MODIFIED qualifier
 - EXCHANGE/NETWORK command, E-19
- Monitor Utility (MONITOR), B-95
 - cluster performance, D-3
 - DECdtm services support, B-95 to B-100
 - MONITOR TRANSACTION command, B-95
 - MONITOR VECTOR command, B-100
 - support for DECdtm services, B-40
 - TRANSACTION class, B-95
 - TRANSACTION class record, B-99
 - VECTOR class, B-100
 - VECTOR class record, B-104
 - vector processing support, B-16
- MONITOR VECTOR command, B-100
- MOUNT command
 - /MEDIA_FORMAT qualifier, B-41
- Mounting of queue file disk, 5-4
- Movefile subfunction
 - calling, 22-1
 - description, 22-1
- /MOVE qualifier
 - in SET FILE DCL command, 11-2
- Moving queue files
 - after queuing system upgrade, 5-4
 - master file, 5-4
 - queue and journal files, 5-4
- MSCP server
 - load balancing, B-55
- MSCP_LOAD parameter
 - using to control load balancing, B-55
- MSCP_SERVE_ALL parameter
 - using to control load balancing, B-55
- Multibuffer count XABITM
 - for increased local buffering, B-110
 - precedence over RAB\$B_MBF field, B-110
- Multiscreen support, B-45
- Multithread program
 - debugger support, 14-1

N

- Name
 - DNS
 - See DNS name
- Name service
 - See DNS (Distributed Name Service)
- Namespace, C-4
 - changing default, C-73
 - clearinghouses in, C-13
 - distributing, C-13
 - listing information, C-20 to C-23
 - name of, C-7, C-42
 - structure of, C-5
 - ways of using, C-4
- /NAMES qualifier
 - for MACRO DCL command, 19-2

- NCP executor, C-1
 - SET/DEFINE EXECUTOR command, C-1
 - SHOW EXECUTOR CHARACTERISTICS command, C-2
- NCR 53C94 controller
 - programming support, A-51
- NETCONFIG.COM command procedure
 - security enhancements, D-2
- NETCONFIG_UPDATE.COM, D-4
- Network Control Program (NCP)
 - line and circuit support for new Ethernet/820 controllers, B-105
 - line and circuit support for VAXft 3000, B-105
- Network default access
 - controlling access to your system, D-2
 - for existing systems, D-4
 - for VAXcluster members, D-4
- /NEW_VERSION qualifier
 - to START/QUEUE/MANAGER command, 5-5
- NEXT subcommand
 - with LMCP REPAIR command, B-92
- /NOCOMMSYNC qualifier
 - in SET TERMINAL DCL command, 18-2
- /NOCONFIRM qualifier
 - EXCHANGE/NETWORK command, E-18
- Node name
 - converting to address, C-60
- /NODE qualifier
 - Error Log Utility (ERROR LOG), B-62 to B-64
- /NOLOG qualifier
 - EXCHANGE/NETWORK command, E-19
- /NOMOVE qualifier
 - in SET FILE DCL command, 11-2

O

- Object
 - See DNS object
- Obsolete command, 5-7
- Obsolete qualifiers, 5-7
- Obsolete queue file, 5-2
- Online help
 - for system messages, B-44
- Opaque name
 - concatenating, C-50, C-54
 - converting to string, C-30, C-33, C-62
 - converting user name, C-64
 - counting components, C-56
 - format of, C-5
 - returning simple name, C-69, C-71
- Open-bus device support
 - SCSI controller, A-51
- Open-bus driver support, A-1, A-8
- Operating system routines
 - for VME drivers, A-21
- Orientation radio box processing option, B-47
- Outgoing connections

- Outgoing connections (Cont.)
 - enabling in LAT, 9-5, 9-7

P

- Page file
 - controlling size in AUTOGEN, B-51
 - deinstalling, D-1
- Page size
 - specifying in link operation, B-108
- Participant in a transaction, B-36, B-39, B-74
- /PARTICIPANTS qualifier
 - in SHOW PROCESS SDA command, 20-3
- Password
 - screening, B-64
 - password history list, B-64
 - site-specific filter, B-65
 - specifying an encryption algorithm, B-65
- Patch Utility (PATCH)
 - support for vectorized programs, B-28
- PEDRIVER data structures, A-62
 - BUS, A-62
 - channel (CH), A-62
 - PORT, A-62
 - port descriptor table (PDT), A-62
 - virtual circuit (VC), A-62
- Phone Utility (PHONE)
 - default access, D-2
- PIO
 - See Programmed I/O
- PIO map registers
 - for VME, A-28, A-29, A-31
- Porting
 - VME device drivers, A-16
- PostScript files
 - CDA Viewer support, B-46
 - VIEW command support, B-46
- Preferred access path
 - programming examples for, A-60
- Preventing autostart queues from starting, 5-9
- Privileges
 - for LMCP commands, B-75
- Proactive memory reclamation, A-1, A-5
- Processing options
 - CDA Viewer
 - orientation radio box, B-47
 - Scale Factor, B-47
 - Use Bitmap Widths toggle button, B-47
 - Use Comments toggle button, B-47
 - Use Fake Trays toggle button, B-48
 - Watch Progress toggle button, B-48
- Process-permanent files
 - VMS RMS asynchronous support, B-110
- Programmed I/O
 - VMEbus device, A-13
- Programming
 - NCR 53C94 controller, A-51
 - VMEbus device driver, A-8

PSWRAP command, B-41

Q

QMAN\$MASTER.DAT, 5-2

changing location after upgrade, 5-4

QMAN\$MASTER logical name, 5-4

defining in a VAXcluster environment, 5-4

Qualifiers

obsolete, 5-7

Queue database

See also Queue files

new design, 5-2

Queue failover, 5-7

Queue files, 5-2

changing location after upgrade, 5-4

location, 5-4

mounting of disk holding, 5-4

moving after upgrade, 5-4

new, 5-2

obsolete, 5-2

Queue manager

autostart, 5-6

availability, 5-1

clusterwide, 5-1

failover, 5-1

function, 5-1

restarting after moving queue files, 5-4, 5-5

separation from job controller, 5-1

starting, 5-1

starting new, 5-5

stopping, 5-7

stopping before moving queue files, 5-4

Queues

availability, 5-7, 5-8

designating autostart, 5-8

failover, 5-8

starting autostart, 5-9

stopping on a node, 5-7

R

RAB\$_MBF field

limitation, B-110

Record blocking

volume mount, B-43

REPAIR command

in LMCP Utility, B-85

ABORT subcommand, B-87

COMMIT subcommand, B-88

EXIT subcommand, B-89

FORGET subcommand, B-90

HELP subcommand, B-91

NEXT subcommand, B-92

Requirements

defining logical name in a VAXcluster environment, 5-5

location of queue and journal file, 5-4

Resource manager, B-36

Restarting queue manager

after moving queue files, 5-4, 5-5

RMS\$_XAB error, B-112

RMS Journaling

support for DECdtm services, B-114 to B-119

RMS services

using XAB\$_NORECORD XABITM, B-112

RTL (Run-Time Library)

DNS\$ routines, C-49 to C-73

LIB\$GETQUI, 13-1

MTH\$ routines, B-7, B-21, B-23

Parallel Processing, 13-3

PPL\$, 13-3

PPL\$DECREMENT_SEMAPHORE, 13-4

PPL\$REMOVE_WORK_ITEM, 13-4

PPL\$UNIQUE_NAME, 13-3

PPL\$WAIT_AT_BARRIER, 13-4

vectorized MTH\$ routines, B-7, B-21, B-23

S

Scalar

defined, B-5

processor synchronization, B-32

Scalar consumer, B-8

Scale Factor processing option, B-47

Screen

supporting more than one, B-45

SCSI data structures

changes, A-51

SCSI device support

NCR 53C94 controller, A-51

SCSI disk class driver

disabling the loading of, B-56

SCSI macro

changes, A-51

SCSI tape class driver

disabling the loading of, B-56

SCSI_NOAUTO system parameter, B-56

Second Generation Ethernet Controller (SGEC)

circuit name, B-105

line name, B-105

Security

enhancements to NETCONFIG.COM

for existing systems, D-4

for new systems, D-2

screening new passwords, B-64

password history list, B-64

site-specific filter, B-65

site-defined password policy, B-64 to B-66

specifying an encryption algorithm, B-65

Separation of job controller and queue manager, 5-1

Service

defined, 9-5

Service announcements, 9-7

Service node, 9-7

Service node (Cont.)

- defined, 9-5, 9-7

Session language

- new languages, B-45
- setting another, B-45

SET ACL command, B-14

SET/DEFINE EXECUTOR command, C-1

SET FILE/MOVE[NOMOVE] command, 11-2

SET HOST/LAT command, 9-5

SET MAGTAPE/RETENSION command, 2-5

SET PREFIX command, 3-6 to 3-7

SET TERMINAL command

- /COMMSYNC qualifier, 18-2

SET VOLUME command

- /REBUILD=FORCE option, 2-4

Shadowing

- See Volume shadowing

SHADOW_MBR_TMO parameter, 8-1

SHOW ACL command, B-14

SHOW/BI=BIindex command

- in System Generation Utility (SYSGEN), B-59

SHOW/BUS=busId command

- in System Generation Utility (SYSGEN), B-60

SHOW command

- LMCP Utility, B-93

SHOW CPU command

- vector processing support, B-15

SHOW ENTRY command, 3-1

- change in format of, 3-1

- executing, 3-2

- jobnames parameter, 3-1

- job state, 3-2

SHOW EXECUTOR CHARACTERISTICS

- command, C-2

SHOW LOGS SDA command, 20-2

SHOW PROCESS command

- vector processing support, B-15

SHOW PROCESS/IMAGES

- SDA (System Dump Analyzer), B-113

SHOW PROCESS/PARTICIPANTS

- SDA (System Dump Analyzer), 20-3

SHOW PROCESS/VECTOR_REGISTERS

- SDA (System Dump Analyzer), B-113

SHOW QUEUE command, 3-2

- change in display, 3-2

SHOW TRANSACTIONS SDA command, 20-6

SHOW/XMI=BIindex command

- in System Generation Utility (SYSGEN), B-61

SHOW ZONE command, B-41

Shutting down queue manager

- before moving queue files, 5-4

Simple name

- converting to opaque, C-31

Simple_Name_Str identifier, C-42

/SINCE qualifier

- EXCHANGE/NETWORK command, E-19

Skulk, C-16

SMP_CPUS parameter, B-11

Soft link

- DNS, C-6

- enumerating, C-29

- locating target entry, C-32

SPI\$CONNECT macro

- using byte count, A-51

SS\$_ACCVIO, B-28, B-29

SS\$_BADCONTEXT, B-31

SS\$_CPUNOTACT, B-31

SS\$_EXQUOTA, B-31

SS\$_ILLVECOP, B-29

SS\$_INSFMEM, B-31

SS\$_INSFWSL, B-31

SS\$_IVADDR, A-5

SS\$_MCHECK, B-31

SS\$_NOPRIV, B-31

SS\$_VARITH, B-28, B-30

SS\$_VASFUL, B-31

SS\$_VECALIGN, B-28, B-30

SS\$_VECDIS, B-30

Stalled job state, 3-2

START/CPU command, B-11

Starting autostart queues, 5-7, 5-9

Starting the LAT protocol software

- with LAT\$STARTUP.COM, 9-1, 9-13

Starting the new queue manager, 5-5

START/QUEUE command

- /AUTOSTART_ON qualifier, 5-8

START/QUEUE/MANAGER command, 5-1

- caution about /NEW_VERSION qualifier, 5-5

- obsolete qualifiers, 5-7

- storage of, 5-6

Startup

- mounting of queue file disk, 5-4

START/ZONE command, B-41

Status

- job, 3-2

STOP/CPU command, B-11

Stopping queue manager

- before moving queue files, 5-4

Stopping queues on a node, 5-7

Stopping the queue manager, 5-7

STOP/QUEUE command

- /ON_NODE qualifier, 5-7

STOP/QUEUE/MANAGER command

- /CLUSTER command, 5-7

STOP/QUEUE/NEXT command

- with autostart queues, 5-9

STOP/QUEUE/RESET command

- with autostart queues, 5-9

STOP/ZONE command, B-41

Stored semantics file attribute

- See File tag

SUBMIT command

- /NOTE qualifier, 3-8

Swap file

- controlling size in AUTOGEN, B-51

- deinstalling, D-1

- SWAPLONG macro, A-19
- Swapping
 - long-waiting processes, A-6
- Swapping bytes, A-19, A-20
- SWAPWORD macro, A-20
- SYLOGICALS.COM
 - mounting queue file disk, 5-4
- Symmetric vector processing configuration, B-5
- Synchronization
 - exception, B-32
 - memory, B-32
- SYNDROME keyword
 - Error Log Utility (ERROR LOG), B-62
- SYS\$DECDTM_INHIBIT logical name, B-34, B-69
- SYS\$DNS system service
 - See \$DNS system service
- SYS\$GETJPI, B-24
- SYS\$GETQUI, 12-1
- SYS\$GETSYI, B-24
- SYS\$JOURNAL logical name, B-67, B-73
 - defining as a search list, B-67
- SYS\$QUEUE_MANAGER.QMAN\$JOURNAL, 5-2
 - changing location after upgrade, 5-4
- SYS\$QUEUE_MANAGER.QMAN\$QUEUES, 5-2
 - changing location after upgrade, 5-4
- SYS\$RELEASE_VP, B-24
- SYS\$RESTORE_VP_EXCEPTION, B-26
- SYS\$RESTORE_VP_STATE, B-26
- SYS\$SAVE_VP_EXCEPTION, B-26
- SYS\$SNDJBC, 12-1
- System disk
 - shadowing the, B-106
- System Dump Analyzer (SDA) commands, 20-1
 - SHOW LOGS, 20-2
 - SHOW PROCESS
 - /PARTICIPANTS qualifier, 20-3
 - /TRANSACTIONS qualifier, 20-4
 - SHOW TRANSACTIONS, 20-6
- System Dump Analyzer (SDA) Utility, A-2
 - modifications for DECdtm services, 16-2
 - PEDRIVER data structures, A-62
 - SHOW PORTS command, A-62
 - support for vectorized programs, B-27
 - vector processing support, B-113
- System Generation Utility (SYSGEN), B-11, B-13, D-1 to D-2
 - commands
 - SHOW/BI=BIindex, B-59
 - SHOW/BUS=busId, B-60
 - SHOW/XMI=BIindex, B-61
 - DEINSTALL command, D-1
 - ERLBUFFERPAGES parameter, D-2
 - increase in lock manager limit values, C-1
 - parameters
 - LOAD_PWD_POLICY, B-57
 - LOAD_SYS_IMAGES, B-57

- System Generation Utility (SYSGEN)
 - parameters (Cont.)
 - SHADOW_MBR_TMO, 8-1
 - TAPE_ALLOCLASS, 7-3
 - TMSCP_LOAD, 7-1
 - using parameters to control load balancing, B-55
- System messages, 4-2
 - accessing with online help, B-44
 - decompressing help library, B-44
 - facilities with new or modified messages, 4-1
- System object
 - default access for, D-2
- System parameters
 - description, D-2
 - displaying
 - bus adapter, B-59, B-60, B-61
- System service, C-23
 - transaction management services, B-39
- System services
 - SYS\$GETQUI and SYS\$SNDJBC, 12-1
- System startup
 - mounting of queue file disk, 5-4
- System tuning
 - automated technique for running AUTOGEN, B-52

T

- Tape
 - cluster-accessible, 7-1
- Tape mass storage control protocol (TMSCP)
 - server, 7-1
- Tape server, 20-1
- Tape support
 - new, A-8
- TAPE_ALLOCLASS parameter, 7-1, 7-3
- Tasking (multithread) program
 - debugger support, 14-1
- TASK object
 - restricting default access, D-2
- Terminal server, 9-8
 - defined, 9-5
- Thread
 - debugger support, 14-1
- Timeout, shadow set member (SHADOW_MBR_TMO), 8-1
- Time-stamps, 3-5
- TMSCP SDA symbol, 20-1
- TMSCP server, 7-1
- TMSCP server code
 - base address, 20-1
- TMSCP_LOAD parameter, 7-1
- TP_SERVER process
 - disabling, B-34, B-69
- Transaction
 - aborting, B-36, B-40
 - abort reason codes, 16-1

Transaction (Cont.)

- atomic, B-35
- committing, B-36, B-40
- examples, B-34
- forgetting, B-90
- monitoring, B-95, B-100
- participants, B-36, B-39, B-74
- states, B-39, B-74
- timeouts, 16-2
- Transaction identifier (TID), B-39, B-74
- Transaction log file, B-38, B-66
 - creating, B-69, B-77, B-78
 - determining location, B-67
 - dumping, B-80
 - estimating file size, B-69
 - format
 - description, B-73
 - sample display, B-74
 - placing in alternate location, B-73
 - repairing, B-85
 - resizing, B-72
 - showing, B-93
- Transaction manager, B-36
- Transaction processing, B-34
- /TRANSACTIONS qualifier
 - in SHOW PROCESS SDA command, 20-4
- Transaction states, B-86
- TRANSACTION_ID data type, B-40
- Transfer modes
 - EXCHANGE/NETWORK command, E-15
- /TRANSFER_MODE qualifier
 - EXCHANGE/NETWORK command, E-19
- Trimming, A-7
- Two-phase commit protocol, B-35, B-39
- TZK10 tape cartridge drive, 2-5

U

- UBMAPEXCED bugcheck, A-25, A-30
- UETP (User Environment Test Package)
 - testing the DECnet connection, D-2
- Use Bitmap Widths toggle button, B-47
- Use Comments toggle button, B-47
- Use Fake Trays Toggle button, B-48
- User Environment Test Package
 - See UETP
- User-specified job retention
 - PRINT/RETAIN command, 3-3
 - SET ENTRY/RETAIN command, 3-3
 - SUBMIT/RETAIN command, 3-3
- User-written programs and procedures
 - default access for, D-2

V

- VAX Ada Run-Time Library, A-2, A-60
- VAXcluster
 - MSCP server load balancing, B-55
 - using multiple CI interfaces, B-55
 - using multiple STAR couplers, B-55
 - volume shadowing in, B-106
- VAXcluster environment
 - defining QMAN\$MASTER in, 5-4
 - queue manager in, 5-1
- VAXcluster failover, B-67, B-68
- VAXft 3000 computer
 - adding a zone to a running system, B-41
 - device names, B-57
 - device types supported by Error Log Utility, B-62
 - displaying current state of system, B-41
 - line and circuit support within NCP, B-105
 - removing a zone from a running system, B-41
 - SHOW ZONE command, B-41
 - START/ZONE command, B-41
 - STOP/ZONE command, B-41
- VAX Procedure Calling Standard
 - requirements for vectorized programs, B-31 to B-33
- VAX Vector Instruction Emulation Facility
 - See VVIEF
- Vector
 - defined, B-5
- Vector arithmetic exception
 - debugging, B-19, B-28
- Vector capability, B-8
 - determining availability within a system, B-15
 - placing an ACL on, B-13 to B-14
- Vector-capable system, B-5
- Vector consumer, B-8
 - determining the identity of, B-14, B-24
 - managing, B-12 to B-14
 - marginal, B-9, B-13
 - obtaining information about, B-14 to B-16, B-24
- Vector context, B-8
 - preserving, B-25, B-32
- Vector context switch
 - fast, B-10
 - obtaining information about, B-14, B-24
 - slow, B-10
- Vector count register, B-5
- Vector CPU time
 - definition, B-15
 - obtaining information
 - about image, B-15
 - about process, B-14, B-15, B-24
 - about processor, B-16
 - about system, B-16
- Vector exception

- Vector exception (Cont.)
 - arithmetic, B-19, B-28
 - memory management, B-28
 - servicing, B-28 to B-31
- Vector exception state
 - preserving across procedure boundaries, B-25 to B-26, B-32
- Vectorized program
 - debugging, B-26 to B-31
 - definition, B-7 to B-8
 - requirements when written in VAX MACRO, B-22
 - writing, B-7, B-21 to B-33
- Vectorizing compiler, B-7, B-21
- VECTOR keyword
 - Error Log Utility (ERROR LOG), B-16, B-62
- Vector length register, B-5
- Vector mask register, B-5
- Vector-present processor, B-5
 - adding to system, B-11 to B-12
 - identifying, B-15, B-24
 - removing from system, B-11 to B-12
 - when unavailable, B-12
- Vector processing, B-4 to B-34
 - benefits of, B-7
 - establishing batch queues for, B-13
 - integrated model, B-5
 - management considerations, B-10 to B-21
 - resource requirements, B-12
 - support within Error Log Utility, B-16
 - support within Monitor Utility, B-100 to B-104
 - support within Patch Utility, B-28
 - system descriptions, B-5 to B-6
 - system messages, B-17 to B-21
- Vector processing support code
 - loading, B-8, B-11
- Vector processing system
 - configuring, B-11 to B-12
 - obtaining information about, B-14 to B-16, B-24
 - obtaining number of vector processors in, B-15, B-24
 - performance, B-5
 - tuning, B-12 to B-13
- Vector processor
 - releasing, B-24
- Vector register, B-5
- Vector state
 - definition, B-25
- VECTOR_MARGIN parameter, B-13
- VECTOR_PROC parameter, B-11
- Version number
 - assigning, E-16
- VIEW command
 - PostScript file support, B-46
 - PS input format, B-41
 - viewing PostScript files, B-41
- VMEbus
 - arbitration, A-10
 - hardware environment, A-9
 - interrupts, A-11
 - parameter selection, A-10
 - programming, A-8
 - protocol, A-10
 - request level, A-10
 - timeout, A-10
- VMEbus device support, A-8
- VME code example
 - DMA interface, A-35
- VME device driver
 - assembling, A-17
 - coding, A-15
 - coding concepts, A-16
 - direct memory access, A-12
 - documentation, A-9
 - interrupt handling, A-11
 - linking, A-17
 - loading, A-17
 - macros, A-18
 - porting, A-16
 - programmed I/O, A-13
 - programming, A-8
 - routines, A-21
 - sample for a DR11-W Emulator, A-35
- VME routines, A-21
- VMSINSTAL
 - deferred running of image, A-1
- VMSINSTAL callback RUN_IMAGE, A-8
- VMS Performance Monitor
 - See VPM
- VMS service node, 9-7
- VMS Volume Shadowing
 - See Volume shadowing, B-106
- Volume shadowing
 - configurations, B-106
 - disk repair and recovery, B-106
 - fault tolerance, B-106
 - in a VAXcluster, B-106
 - mixing phase I and phase II, B-106
 - overview, B-106
 - phase II support, B-106
 - the system disk, B-106
 - types, B-106
- VPM (VMS Performance Monitor), D-3
 - default access for, D-2
- VVIEF\$DINSTAL.COM, B-16
- VVIEF\$INSTAL.COM, B-16
- VVIEF (VAX Vector Instruction Emulation Facility)
 - determining presence of, B-15, B-16, B-24
 - loading, B-16
 - overview, B-10
 - unloading, B-16

W

Watch Progress toggle button, B-48
Wildcard character
 DNS, C-9, C-20
 EXCHANGE/NETWORK command, E-16

X

XAB\$_ENABLE symbol, B-112
XAB\$_MULTIBUFFER_COUNT XABITM
 implementation of, B-110
 supporting data structure requirement, B-110
XAB\$_NORECORD XABITM, B-112
 buffer requirement, B-112
 typical usage, B-112
XMI-to-VME routines, A-21

THE UNIVERSITY OF CHICAGO
LIBRARY
540 EAST 57TH STREET
CHICAGO, ILL. 60637
TEL: 773-936-5000
FAX: 773-936-5001
WWW.CHICAGO.EDU

WILLIAM L. BRYANT
1856-1912
POET, ORATOR, AND
JOURNALIST

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	_____	Local Digital subsidiary or approved distributor
Internal ¹	_____	USASSB Order Processing - WMO/E15 or U.S. Area Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

¹For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Forward Order Additional Documentation

Technical Support

It is our goal to provide you with the best technical support possible. Please contact us if you need assistance with your order or have any questions.

Shipping and Delivery

We strive to get your order to you as quickly as possible. Please allow 3-5 business days for processing and shipping. Delivery times may vary depending on your location.

Terms and Conditions of Sale

Product Description	Quantity	Unit Price	Total Price
Item 1: [Product Name]	1	\$100.00	\$100.00
Item 2: [Product Name]	2	\$50.00	\$100.00
Item 3: [Product Name]	1	\$200.00	\$200.00
Item 4: [Product Name]	3	\$30.00	\$90.00
Item 5: [Product Name]	1	\$150.00	\$150.00
Item 6: [Product Name]	1	\$100.00	\$100.00
Item 7: [Product Name]	1	\$100.00	\$100.00
Item 8: [Product Name]	1	\$100.00	\$100.00
Item 9: [Product Name]	1	\$100.00	\$100.00
Item 10: [Product Name]	1	\$100.00	\$100.00
Item 11: [Product Name]	1	\$100.00	\$100.00
Item 12: [Product Name]	1	\$100.00	\$100.00
Item 13: [Product Name]	1	\$100.00	\$100.00
Item 14: [Product Name]	1	\$100.00	\$100.00
Item 15: [Product Name]	1	\$100.00	\$100.00
Item 16: [Product Name]	1	\$100.00	\$100.00
Item 17: [Product Name]	1	\$100.00	\$100.00
Item 18: [Product Name]	1	\$100.00	\$100.00
Item 19: [Product Name]	1	\$100.00	\$100.00
Item 20: [Product Name]	1	\$100.00	\$100.00

The total price of this order is \$2,000.00. Please allow 3-5 business days for processing and shipping. Delivery times may vary depending on your location.

NOTES

NOTES

NOTES

NOTES

Reader's Comments

VMS Version 5.5 New Features
Manual

AA-LA97D-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:

	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

I would like to see more/less _____

What I like best about this manual is _____

What I like least about this manual is _____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

Phone _____

Do Not Tear - Fold Here and Tape

digital™



No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Information Products
ZK01-3/J35
110 SPIT BROOK RD
NASHUA, NH 03062-9987



Do Not Tear - Fold Here

Reader's Comments

VMS Version 5.5 New Features
Manual

AA-LA97D-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:

	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

I would like to see more/less _____

What I like best about this manual is _____

What I like least about this manual is _____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™



No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Information Products
ZK01-3/J35
110 SPIT BROOK RD
NASHUA, NH 03062-9987



Do Not Tear - Fold Here

